الجمهورية الجزائرية الديمقراطية الشعبية

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR
ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE DR. YAHIA FARES DE MEDEA

FACULTE DES DE LA TECHNOLOGIE
Département du Génie Électrique



Projet de Fin d'études

Présenté pour l'obtention du Diplôme
de Master en Télécommunications

Spécialité : **Systèmes Intelligents et Communicants**

THEME

CONTRIBUTION A L'OPTIMISATION DU TEMPS D'ENCODAGE POUR LA NORME HEVC

Proposé et dirigé par,
M. MAAZOUZ Mohamed
M. TCHOKETCH-KEBIR Sofiane

Préparé et présenté par,
Melle. BOUFARES Sarra
Melle. ZEKRI Meriem

AnnéeUniversitaire 2016-2017

Dédicace

Je tiens à remercier en premier lieu dieu le tout puissant qui a Éclairé mon chemin pour achever ce travail.

A mes très chers parents, que ce modeste travail, soit pour vous une petite compensation et reconnaissance envers ce que vous avez fait d'incroyable pour moi.

A mes très chers frères, sœurs.

A mon marie foudil et sa famille.

A mes préfère ami(e)s Méry & Nour el hauda , Oumayma, Fate, Abir, Wassila, Maroua, Hadjer.

En témoignage de l'amitié sincère qui nous a liées et des bons moments passés ensemble. Je vous dédie ce travail en vous souhaitant un avenir radieux et plein de bonnes promesses.

Ou l'encre des savants est bien meilleure auprès de lieu que le sang des martyrs.

Sarra

Dédicace

j'ai immense honneur de dédier ce modeste travail:

A mes chers parents qu'étaient présents pour moi durant toute ma vie.

A mon marie, à celui qu'il m à encouragé beaucoup pour continuer mes études et m'apprenais le but dans ma vie.

A mes frères et mes sœurs.

A ma grands mères et ma tants.

A ma binôme "Sarra".

A toutes mes chers amies, chacune par son propre nom.

Meriem

Remerciements

Nous remercions tout d'abord, ***ALLAH****, le grand* ***DIEU****, qui nous a donné la force et le courage afin de parvenir à élaborer ce modeste travail.*

Nous remercions nos parents, pour tout leur amour, leur encouragement.

Nous tenons à remercier de tout cœur notre encadreur ***Mr. M.MAAZOUZ Mohamed*** *et témoigner toute notre reconnaissance pour ses conseils judicieux, de ses remarques objectives et surtout de ses gentillesses permanentes.*

Nous témoignons de notre gratitude à l'ensemble des enseignants d'université ***Dr.YAHIA-FARES*** *de Médéa.*

Nos vifs remerciements à tous les membres du jury pour avoir accepté d'examiner et d'évaluer ce mémoire.

Nous tenons également a remercier toutes les personnes qui nous ont aidés de prés ou de loin avec leurs conseils.

ملخص:

ان ظهور نظام HEVC (ترميز فيديو بكفاءة عالية) تصاحبه مشاكل جدية تتعلق بالموارد المستهلكة و وقت الترميز.لذا فان اقتراح الأدوات والتحسينات الجديدة يوصى بشدة لضمان تكامل هذا التشفير الجديد مع مختلف المنصات وتطبيقات الوسائط المتعددة . في هذا السياق ، جاء ترميز Kvazaar للتغلب على المشاكل المتعلقة بنموذج (HM). إن هذا البرنامج الأكاديمي المجاني متلائم لاحتياجات المبرمجين من خلال السماح المعالجة المتوازية على مستوى عال.

إننا في هذا المشروع ، نقترح حلين لتقليل وقت الترميز. الأول يشمل البرمجة المتوازية لمنصات متعددة النوى ، في حين أن الثاني يتمثل في تحسين خوارزمي حسابي على مستوى التقسيم لتجنب بعض القطع LCU ومن خلاله تقليل الوقت الترميز. تظهر نتائج التنفيذ ربح يصل الى-69٪ في وقت الترميز مع تدهور ضئيل جدا (الجودة و التدفق).

الكلمات المفتاحية: ترميز الفيديو ، معالجة متوازية ، تحسين الأداء ، تقسيم.

Résumé :

L'émergence de la nouvelle norme HEVC (High Eficiency Video Coding) est accompagnée de graves problèmes liés à la consommation de ressources et au temps d'encodage. La proposition de nouveaux outils et optimisations est fortement recommandé pour assurer l'intégration de ce nouveau codeur sur différentes plates-formes et les applications multimédia. Dans ce contexte, l'encodeur Kvazaar HEVC est introduit pour surmonter les problèmes liés à HEVC logiciel de référence modèle de test (HM). Ce software académique libre est adapté aux besoins du programmeur en permettant un traitement parallèle à haut niveau.

Dans ce projet, nous proposons deux solutions pour réduire le temps d'encodage. La première concerne la programmation parallèle pour les plateformes multi-cœurs, tandis que la deuxième est une optimisation algorithmique au niveau du partitionnement pour éviter certaines formes de blocs LCU en réduisant le temps d'encodage. Les résultats d'implémentation montrent un gain jusqu'à -69% en temps d'encodage pour une dégradation négligeable en performances (qualité et débit).

Mots clés : Codage Vidéo, Implémentation Parallèle, Optimisation, Partitionnement.

Abstract :

The emergence of the new HEVC standard (High Efficiency Video Coding) is accompanied by serious problems related to resource consumption and encoding time. The proposal of new tools and optimizations is strongly recommended to ensure the integration of this new encoder on different platforms and multimedia applications. In this context, the HEVC Kvazaar encoder is introduced to overcome problems related to HEVC software model test model (HM). This free academic software is adapted to the needs of the programmers by allowing parallel processing at high level.

In this project, we propose two solutions to reduce the encoding time. The first concerns parallel programming for multi-core platforms, while the second is an algorithmic optimization at the partitioning level to avoid certain forms of LCU blocks by reducing the encoding time. Implementation results show up to -69% gain in encoding time with negligible performance degradation (quality and bitrate).

Key words: Video Coding, Parallel Implementation, Optimization, Partitioning.

Table des matières

1. Introduction 3
2. Les normes de codage vidéo 4
2.1. ITU (International Telecommunication Union) 4
2.2. ISO (International Organization for Standardization) 4
3. Les standards de codage vidéo 5
4. Structure de l'encodeur HEVC 7
4.1. Description générale 7
4.2. Le partitionnement dans l'encodeur 8
4.2.1. Les blocs de prédiction 8
4.2.2. Les unités de Transformation 9
4.2.3. L'arbre de codage 10
4.3. La transformation dans la norme HEVC 11
4.4. La quantification 12
4.5. La Prédiction Intra 12
4.6. La Prédiction Inter 14
4.7. Le Filtrage adaptatif 17
4.8. Le Codage entropique/ CABAC 18
5. Les formats de vidéo 20
6. Les niveaux de parallélisme pour l'encodeur HEVC 22
6.1. Slice 22
6.2. Tile 24
6.3. Wpp(WavefrontParallelProcessing) 26
6.4. Comparions entre les outils de traitement parallèle 27
7. Transformation des couleurs 29
8. Conclusion 30
1. Introduction : 31
2. Architectures pour le calcul parallèle 31
2.1. Architectures à mémoire partagée 32

2.2. Architectures à mémoire distribuée 33
2.3. Architectures hybrides 33
3. Programmation des machines parallèles 34
3.1. Méthodologie de parallélisation 34
3.1.1. Choix de la granularité des tâches 35
3.1.2. Expression des interactions entre tâches 35
3.1.3. Ordonnancement et placement des tâches 36
3.2. Modèles de programmation parallèle 37
3.2.1. Programmation des architectures à mémoire partagée 38
3.2.2. Programmation des architectures à mémoire distribuée 38
3.2.3. Programmation des grappes de multiprocesseurs 39
3.2.4. Programmation indépendante d'architecture 40
4. Multiprogrammation légère pour le calcul parallèle 41
4.1. Les processus légers en bref 41
4.2. Le bénéfice à tirer des processus légers 42
5. OpenMP (Open Multi-Processing) 43
6. Conclusion 45
1. Introduction 46
2. La détection de contour par les méthodes dérivatives 46
2.1. Rappels sur le filtrage linéaire 47
2.1.1. Les filtres étroits 48
2.1.2. Les filtres larges 51
3. L'algorithme de l'homogénéité 55
4. Conclusion 55
1. Introduction 56
2. Outils et logiciels utilisés 56
3. Implémentation et résultats 57
3.1. L'implémentation de base 57
3.2. Programmation multithreads 58
3.3. Anticipation du partitionnement selon l'homogénéité 60

4. Conclusion 62
Conclusion générale 63
Bibliographie 64

Liste des Tableaux

Tableau 1. Caractéristiques de la plateforme utilisée 56
Tableau 2. Séquences vidéo utilisées 57
Tableau 3. Paramètres de configuration pour l'encodage 57
Tableau 4. Résultats d'implémentation en utilisant 1 seul thread 58
Tableau 5. Résultats d'implémentation en utilisant 2 et 4 Threads 60
Tableau 6. Résultats d'implémentation par optimisation du partitionnement et multithreading 62

Liste des figures

Figure 1.L'historique des différentes normes et standards par le temps. 5
Figure 2. Structure de l'encodeur HEVC 8
Figure 3. Le partitionnement des blocs pour le HEVC 9
Figure 4. (a) Les unités de prédiction symétriques et (b) asymétriques. 9
Figure 5. Un exemple de partitionnement d'une UC 32x32 en TU etTU. 10
Figure 6. Exemple pour le partitionnement d'une unité d'arbre de codage 64x64 (CTU) dans des unités de codage (UC) de 8x8 à 32x32 échantillons de luma 11
Figure 7. Les modes d'intra-prédiction HEVC par rapport à leurs modèles de balayage avec quelques exemples résiduels moyens dans le domaine de transformation pour 4×4 UT 13
Figure 8. Les exemples de résidus de intra-prédiction de 4x4 blocs TU extraits de séquence BQ Square en résolution de 416x240 14
Figure 9. Représentation schématique d'une séquence temporelle d'images 15
Figure 10. Avec la prédiction inter-images, chaque image d'une séquence est classifiée comme un certain type d'image (image I, image P ou image B) 15
Figure 11. Illustration de la compensation de mouvement basée sur bloc 16
Figure 12. Types de prédictions dans un ensemble d'images (Groupe of Pictures).....17
Figure 13. Codage entropique dans HEVC 19
Figure 14. Exemple de format 4 :4 :4 20
Figure 15. Exemple de format 4 :2 :2 21
Figure 16. Exemple de format 4 :2 :0 22
Figure 17. Structure de l'image en HEVC à l'aide de slices, montrant les CTU appartenant à des segments indépendants de slices (bleu ombré) et slice des segments dépendants de deux tranches d'une image.....23
Figure 18. Ordre de balayage de trame basé sur Tiles de CTU avec neuf Tiles de tailles différentes dans l'image. Notez que les limites des Tiles sont indiquées par des tirets gras. ...25
Figure 19. Exemple de partitionnement de Tiles en utilisant deux Tiles en slice et slice de segments. 25
Figure 20. Illustration de WPP avec des threads 26
Figure 21. Partitionnement en sous-images.....27
Figure 22. Construction d'un décodeur multi-core en répliquant des décodeurs à noyau unique 29
Figure 23. Format YUV 4 :1 :1 29

Figure 24. Programmation classique vs. multiprogrammation d'un programme composé de tâches indépendantes 43
Figure 25.Parallélisation d'une région parallèle avec OpenMP 44
Figure 26. Parallélisation des boucles avec OpenMP 44
Figure 27. Parallélisation des sections avec OpenMP 45
Figure 28 Illustration de la notion de contour 47
Figure 29 Le filtrage gradient 49
Figure 30. Masques de Roberts 49
Figure 31. Masques de Prewitt horizontal et vertical 50
Figure 32. Masques de Sobel horizontal, vertical et oblique 50
Figure 33. Masques de Kirsch 51
Figure 34. Le filtrage par dérivée seconde 52
Figure 35. Masques d'approximation du Laplacien 52
Figure 36. Image originale, filtrage de Sobel et filtrage Laplacien 53
Figure 37. Dé-bruitage et détection de contours 53
Figure 38. Laplacien de la fonction de Gauss 54
Figure 39. Masque Laplacien 54
Figure 40. Exemple de calcul de l'homogénéité 55
Figure 41. Organigramme pour la sélection de la profondeur maximale de recherche 61

Introduction générale

La demande de services vidéo de haute qualité implique le développement de nouvelles normes de compression vidéo plus efficaces. Ces derniers devraient résoudre des problèmes concernant le stockage d'énormes quantités d'informations et la bande passante requise pour la transmission. Dans ce contexte, la norme de codage vidéo HEVC est développée. C'est le dernier travail publié par *Joint Collaborative Team on Video Coding* (JCT-VC) en collaboration avec ITU-T*Video Coding Experts Group*(VCEG) et ISO/IEC *Moving Picture Experts Group* (MPEG).Les performances attendues par le codeur HEVC concernent la réduction du débit de transmission à la moitié pour la même qualité par rapport à son réputé prédécesseur H.264/AVC. Cependant, la complexité résultante est le principal défi pour les développeurs. L'encodage avec HEVC en temps réel reste limité à quelque simplémentations ou produits commerciaux. La proposition de nouveaux outils et optimisations est fortement recommandée pour assurer l'intégration de ce nouveau codeur sur différentes plates-formes et applications multimédia

Les progrès remarquables dans le domaine de la micro-électronique fournissent des microprocesseurs toujours plus puissants, à un rythme souvent supérieur aux prévisions les plus optimistes. Seulement cet accroissement en puissance demande des technologies de plus en plus complexes, engendrant des difficultés qui se font remarquer tant sur le plan technologique que sur le plan économique. En même temps, les besoins des utilisateurs du calcul haute-performance sont habituellement au-delà des ressources disponibles. En effet, la conquête de la puissance nécessaire pour effectuer des calculs de pointe est naturellement suivie par le lancement de nouveaux challenges. Cela se traduit par des besoins toujours croissants en termes de performance arithmétique, capacité de stockage et rapidité d'accès aux données.

Le calcul parallèle constitue une réponse aux problèmes soulevés ci-dessus. Il consiste à utiliser simultanément les différentes ressources disponibles dans un système informatique (processeurs, mémoires, disques, réseaux de communication, etc.) pour l'exécution des calculs correspondants à une même application. Les motivations pour une telle technique ont à la fois un caractère temporel (travailler plus vite) et spatial (réaliser des plus gros travaux). De plus, le parallélisme peut être vu comme une façon de contourner les barrières technologiques et économiques qui s'imposent au développement de microprocesseurs plus puissants. L'utilisation de multiples processeurs pour une même application implique une coopération entre eux. Ce fait est à l'origine des grandes difficultés liées au parallélisme : d'un point de vue matériel parce que cela nécessite des technologies permettant un accès

rapide à des informations qui ne sont pas localement accessibles à un processeur, et d'un point de vue logiciel car il faut repenser les outils classiques de programmation afin de prendre en compte la coopération entre processeurs.

Dans ce mémoire de fin d'études, deux types d'optimisations sont proposés comme solutions pour l'encodeur HEVC. La première technique d'optimisation concerne l'exploitation des plateformes multiprocesseurs pour le traitement parallèle des données, ce qui permet de réduire considérablement le temps d'encodage. Tandis que la deuxième proposition opère même sur des plateformes monoprocesseurs. C'est une optimisation algorithmique qui utilise les propriétés intrinsèques d'un bloc (zone de l'image) pour éviter certains calculs onéreux.

Le manuscrit de ce projet est organisé en quatre parties. La première partie présente des concepts fondamentaux sur les normes de codage vidéo, en particulier l'encodeur HEVC. Le chapitre 2 présente quelques outils et techniques de programmation parallèle. Le troisième chapitre discute de la détection de contour, en présentant quelques méthodes de segmentation dans le but d'une classification des blocs Homogène/Non-homogène. Les résultats expérimentaux avec discussions sont présentés dans le quatrième chapitre. Enfin, le manuscrit est clôturé par une conclusion générale et quelques perspectives.

Chapitre I : Structure de l'encodeur HEVC

1. Introduction

La vidéo numérique est devenue omniprésente dans notre vie de tous les jours; Partout où nous regardons, il y a des dispositifs qui peuvent afficher, capturer et transmettre des vidéos. Les récentes avancées technologiques ont permis de capturer et d'afficher du matériel vidéo avec une résolution ultra-haute définition[1] (UHD).

La nécessité d'un système de transmission amélioré est plus prononcée dans le secteur mobile en raison de l'introduction de résolutions HD légères (telles que 720 pixels) pour les applications mobiles.

Les limites des technologies actuelles ont incité l'organisation internationale de normalisation Moving Picture Experts Groupe (MPEG) et l'Union Internationale des Télécommunications (UIT) pour établir le Teamon Video Coding (JCT-VC), dans le but de développer une nouvelle norme de codage vidéo haute performance. Un appel formel à propositions sur la technologie de compression vidéo a été lancé en janvier 2010, et 27 propositions ont été reçues en réponse à cet appel. Ces propositions ont été présentées lors de la première réunion du JCT-CV en avril 2010.

Les évaluations qui ont suivi ont montré que certaines propositions pouvaient atteindre la même qualité visuelle que le Codage Vidéo Avancé[3] (AVC) H.264 / MPEG-4 à seulement la moitié du débit binaire et au coût de deux à dix fois l'augmentation du calcul complexité.

Certaines autres propositions pourraient atteindre une bonne qualité subjective et des débits binaires avec une complexité informatique plus faible que l'encodeur AVC de haut niveau de référence.

Depuis lors, JCT-VC a mis un effort considérable en vue du développement d'une nouvelle norme de compression connue sous le nom de norme HEVC (High Efficiency Video Coding), dans le but d'améliorer considérablement l'efficacité de la compression par rapport aux H.264 / AVC existants haut profil.

Ce chapitre fournit une vue d'ensemble de HEVC tel qu'il est actuellement, examiner ses principales caractéristiques.

2. Les normes de codage vidéo

2.1.ITU (International Telecommunication Union)

L'UIT est une agence spécialisée des Nations Unies (UN)en 1947 basée à Genève (Suisse), qui est responsable des problèmes qui concernent les technologies de l'information et de la communication. C'est une organisation intergouvernementale de partenariat public-privé. Elle est composée de 193 pays et environ 700 entreprises publiques et privées ainsi que des entités de télécommunication internationales et régionales, qui assurent la majeure partie du travail de chaque secteur, qui est responsable des problèmes concernant les technologies de l'information et de la communication et améliore les infrastructures de télécommunication dans les pays en développement et aide à l'élaboration et à la coordination de normes techniques mondiales.

L'UIT est active dans des domaines tels que l'Internet à large bande, les technologies sans fil de dernière génération, la navigation aéronautique et maritime, la radioastronomie, la météorologie par satellite, la convergence des téléphones fixes, l'accès à internet, les données, la voix, la télé diffusion, etc.

L'UIT comprend trois secteurs[4], chacun gérant un aspect différent des questions traitées par l'union, ainsi que l'UIT Telecom. Les secteurs ont été créés lors de la restructuration de l'UIT lors de sa Conférence de plénipotentiaires de 1992 :

- Communication radio (UIT-R).
- Normalisation (UIT-T).
- Développement (UIT-D).

2.2.ISO (International Organization for Standardization)

Le nom ISO dérive du mot grec iso, qui signifie égal, fondée en 1946, c'est une organisation internationale composée d'organismes nationaux de normalisation de plus de 75 pays. Par exemple, l'ANSI (American National Standards Institute) est membre de l'ISO.

L'ISO a défini un certain nombre de normes informatiques importantes et dans de nombreuses entreprises et de technologies, les ordinateurs et les communications, dont la plus importante est peut-être l'OSI (Open System Interconnexion) [5], une architecture normalisée pour la conception de réseaux.

L'ISO a publié plus de 21000 Normes internationales qui peuvent être achetées auprès de l'ISO ou de ses membres pour assurer l'aptitude à l'emploi des matériaux, produits, processus et services. Par exemple, Les normes ISO les plus connues :

ISO/IEC 27001 – Management de la sécurité de l'information

ISO 50001 - Management de l'énergie

ISO 45001 – Santé et sécurité au travail

ISO 14000 - Management environnemental.

3. Les standards de codage vidéo

Une série des normes et standards de codage vidéo pour but de présenter le travail des Groupes d'action du MPEG (1988)(ISO/IEC Moving Picture Expert Group) et VCEG (1997) (ITU Video coding Expert Group)[6] et Joint Video Team: H.264 et MPEG-4/Part 10 (JVT) Équipe vidéo commune entre ISO et ITU, afin de mieux comprendre l'historique des différentes normes et standards.

La figure 1 présenté les différents standards et normes par le temps :

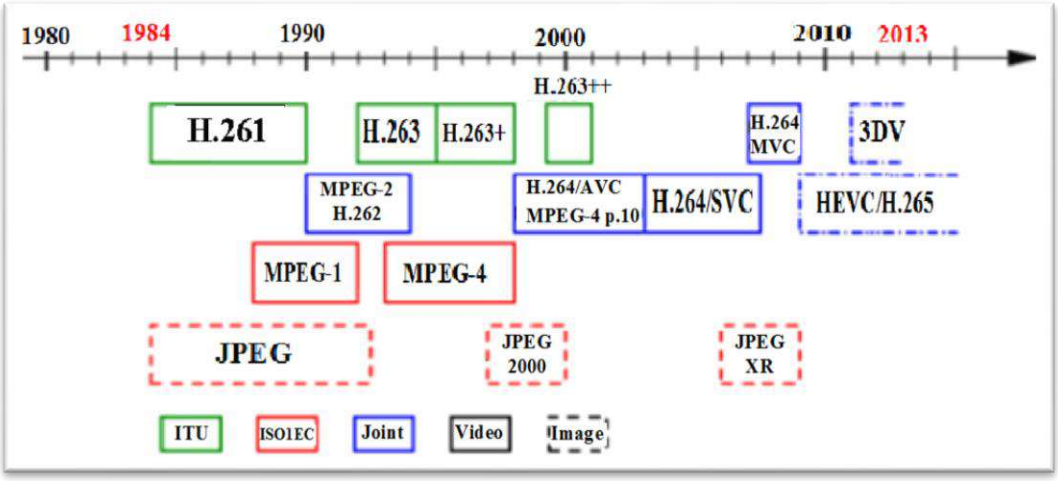


Figure 1.*L'historique des différentes normes et standards par le temps.*

Les différents standards pour UIT sont :

a. H.120 : est le premier standard d'encodage de la vidéo numérique. Il a été développé par COST 211 et publié par le CCITT en 1984, une première révision est parue en 1988, et une dernière en 1993. La vidéo obtenue n'avait pas une qualité convenable, peu d'implémentations réelles ont été faites[1].

b. H.262 : est le nom de la recommandation UIT-T correspondant à la norme ISO/CEI 13818-2, c'est-à-dire MPEG-2 Vidéo. Les textes des deux normes sont identiques[1].

c. H.263 : est une recommandation concernant la norme de codage vidéo développé en février1995 par l'UIT-TQ.6/SG16. Il a été dédié aux communications à faible débit (moins de 64 Kbit/s).Elle a été développée pour la transmission de la vidéo sur des lignes à très bas débits [2].

d. H.264 : ou MPEG-4 AVC (Advanced Vidéo Coding), ou MPEG-4 Part 10, est une norme de codage vidéo développée conjointement par l'UIT-T Q.6/SG16 Vidéo Coding Experts Group (VCEG) ainsi que l'ISO/CEIMoving Picture Experts Group (MPEG) et est le produit d'un effort de partenariat connu sous le nom Joint Vidéo Team (JVT). La norme UIT-T H.264 et la norme ISO/CEI MPEG-4 Part 10 (ISO/CEI 14496-10) sont techniquement identiques, et la technologie employée est aussi connue sous le nom AVC, pour Advanced Vidéo Coding[1].
e. H.265, ou « MPEG-4 HEVC » (High Efficiency Vidéo Coding), est une norme de codage vidéo ISO/CEI 23008-2 et UIT-T H.265, publiée le 13 avril 2013. Elle est développée conjointement par les groupes Vidéo Coding Experts Group (VCEG) et Moving Picture Experts Group et doit succéder au H.264 (ISO/CEI 14496-10 et UIT-T H.264)[1].

Les différentes normes pour ISO sont :

a. MPEG-1 : Le descendant du Mjpeg [3]: Développé entre 1988-1992 par ISO sous le code ISO/IEC 11172, a été finalisée par le groupe MPEG[11].C'est leur premier standard audio et vidéo utilisé plus tard comme standard des Vidéo CDs à un débit inférieur à 1.86 Mbps (qualité VHS) [2].
b. MPEG -2 : La diffusion en numérique[3]: Développé entre 1990-1994par ISO sous le code ISO/IEC 13818[2] et UIT (H.262) [4]. Elle a été développée pour la compression de la vidéo de qualité télévision à un débit de 4 à 6 Mbits/s (ce qui correspond à un canal de télévision normal). Un peu plus tard, MPEG-2 intégra la compression de la télévision à haute définition (HDTV). Aujourd'hui, MPEG-2 est aussi le format utilisé pour stocker les films sur DVD [2].
c. MPEG -4 : Le Multimédia par excellence [3]: C'est en 1994 que MPEG (Moving Picture Experts Group) a développé MPEG-4. Le protocole est devenu un standard officiel ISO/IEC en janvier 1999. Il vise à fusionner trois domaines : l'informatique, la télécommunication et la télévision à un débit vidéo de10 Kbp/s à 1 Gbp/s [4]. La norme MPEG-4 est radicalement différente des normes précédentes. Elle initialise une vidéo orientée objets : une séquence est une scène hiérarchisée en objets média, chaque objet est composé d'un ensemble de paramètres (son, forme, position,. . .) [2].
    - La Partie 2 "Visual" implique des fonctionnalités en plus telles que l'interactivité sur le contenu audiovisuel, mais n'a pas vraiment connue de succès (trop d'options et de possibilités différentes).

- La Partie 10 "AVC" (Advanced VideoCoding) se recentre sur l'efficacité de codage de façon impressionnante (débit 2 fois plus faible pour qualité visuelle comparable) et abandonne "le reste" (pas d'objets, pas d'interactivité).

d. MPEG-7 : Gérer une bibliothèque Multimédia : Introduite en 1997défini des outils capable de décrire de façon standardisée un contenu multimédia afin de rendre rapide et efficace des recherches et des classifications de documents.

e. MPEG-21: Le business et la protection juridique : vise quant à lui la commercialisation automatisée et contrôlée de tout type de document multimédia, MPEG-21 est ratifié dans les normes ISO / CEI 21000 - Cadre multimédia (MPEG-21).

4. Structure de l'encodeur HEVC

4.1.Description générale

La compression vidéo est une méthode de compression de données, qui consiste à réduire la quantité de données, en minimisant l'impact sur la qualité visuelle de la vidéo[2]. L'intérêt de la compression vidéo est de réduire les coûts de stockage et de transmission des fichiers vidéo. Différentes technologies de compression, représentant des normes à la fois propriétaires et industrielles, sont disponibles [5].

HEVC (High Efficiency Video Coding) Standard, est conçu selon le principe du codage vidéo hybride basé sur des blocs pour la transmission. La Figure 2Montre un schéma synoptique d'un encodeur vidéo hybride à base de blocs avec certains ingrédients caractéristiques de HEVC concernant son concept de partitionnement de blocs.

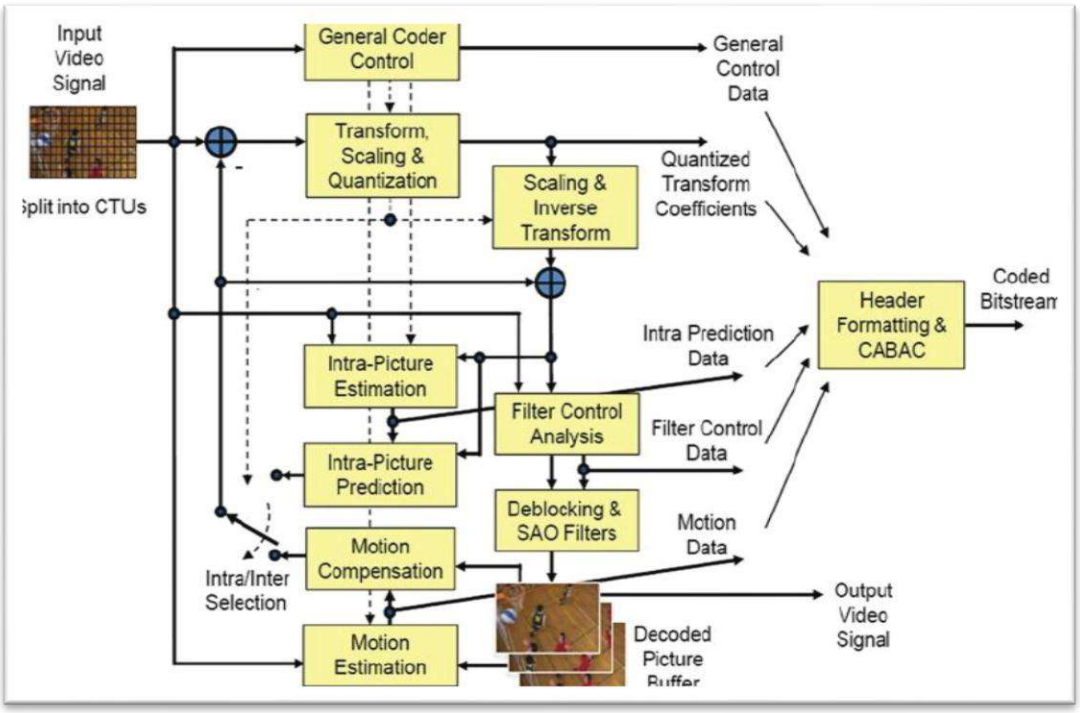


Figure 2. *Structure de l'encodeur HEVC*

4.2.Le partitionnement dans l'encodeur

Semblable aux normes de codage vidéo classiques, HEVC est un schéma de codage hybride à base de blocs. L'un des principaux facteurs contribuant à la performance de compression plus élevée de HEVC est l'introduction de structures de blocs plus larges avec des mécanismes de subdivision flexibles.

Le bloc de base dans HEVC est connu comme la plus grande unité de codage(Large coding unit) (LCU), Une CTU est une forme carrée avec une taille de 8x8 à 64x64 pixels, La notion de LCU dans HEVC est généralement similaire à celle d'un macrobloc dans les normes de codage précédentes. En utilisant le partitionnement récursif quad-arbre et peut être récursivement divisé en unités de codage plus petites (UC), la taille d'un bloc CU peut être 64x64, 32x32, 16x16, 8x8, Chaque UC est codée par prédiction unités (PU) en mode intra ou inter-prédiction. La Figure 3 montre un exemple de partitionnement d'un LCU 64x64 à différentes tailles d'UC.

4.2.1. Les blocs de prédiction

Chaque UC peut être divisée en unités plus petites, qui forment la base de la prédiction. Ces unités sont appelées PU. Chaque CU peut contenir une ou plusieurs PUS, et chaque PU peut être aussi grand que leur CU racine ou aussi petite que 4x4 dans les tailles de bloc luma. Alors qu'un LCU peut se diviser de façon récursive en UC plus petites et plus

petites, le fractionnement d'une CU en PUS est non récursif (il ne peut être effectué qu'une seule fois). Les PU peuvent être symétriques ou asymétriques. Les PEM symétriques peuvent être carrées ou rectangulaires (non carrées) et sont utilisées à la fois dans l'imprévu (utilise uniquement des PU carrées) et l'inter-prédiction En particulier, une UC de taille 2Nx2N peut être divisée en deux PU symétriques de taille Nx2N ou 2NxN ou quatre PUs de taille NxN.

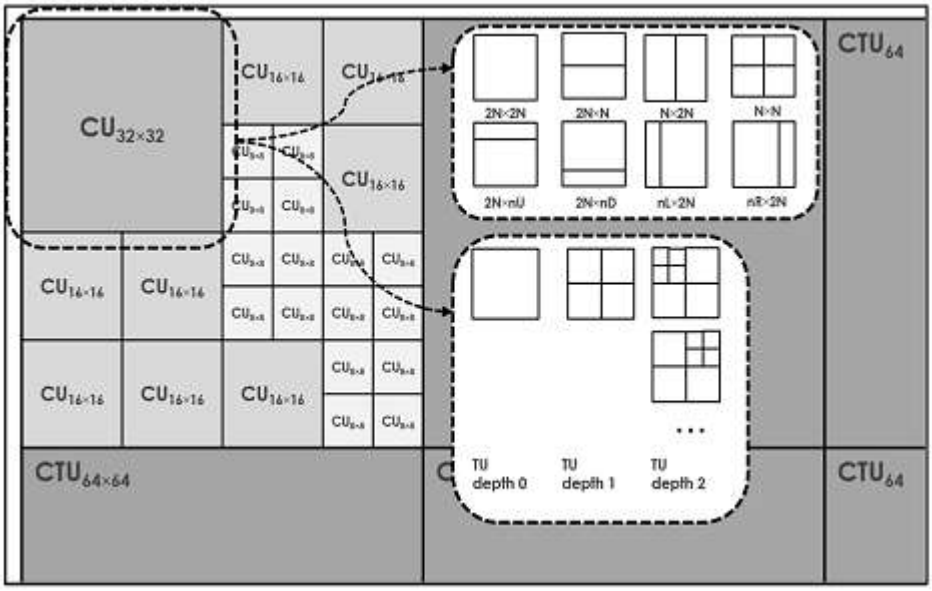


Figure 3. *Le partitionnement des blocs pour le HEVC*

Les PU asymétriques sont utilisés uniquement pour l'inter-prédiction. Cela permet le partitionnement, qui correspond aux limites des objets de l'image. La Figure 4 montre le partitionnement d'une CU sur des PU symétriques et asymétriques.

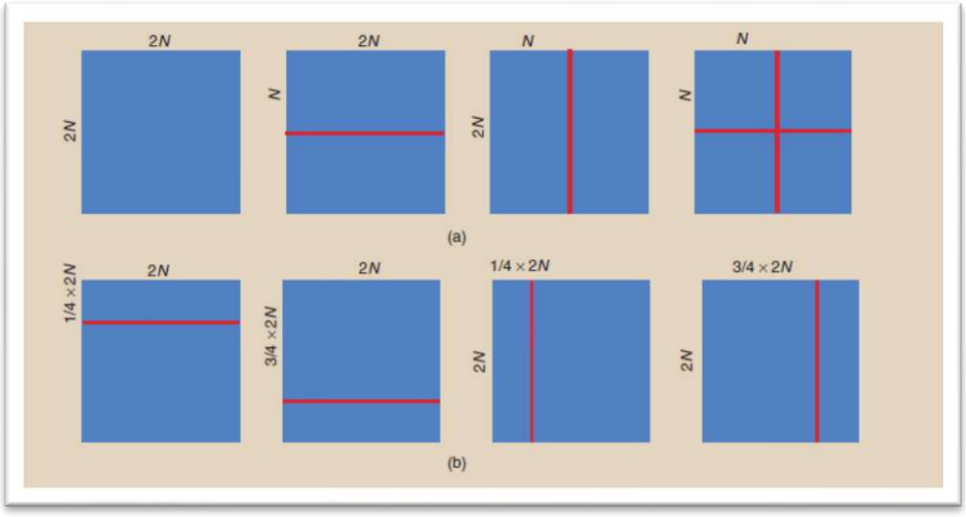


Figure 4. *(a) Les unités de prédiction symétriques et (b) asymétriques.*

4.2.2. Les unités de Transformation

Comme pour les normes de codage vidéo précédentes, HEVC applique une transformée de cosinus discrète (DCT) à des données dé-corrélées. En HEVC, une unité de

transformation (TU) est l'unité de base pour les processus de transformation et de quantification. La taille et la forme de la TU dépends de la taille de la PU[6].

La taille et la forme de la TU dépends de la taille de la PU. La taille des UT de forme carrée peut être aussi petite que 4x4 ou aussi grande que 32x32. Les UT non adjacentes peuvent avoir des tailles de 32x8, 8x32, 16x4 ou 4x16 échantillons de luma. Chaque CU peut contenir une ou plusieurs UT; Chaque UC carrée peut se diviser en plus petites UT dans une structure de segmentation quad-arbre. La Figure 5 montre un exemple de la façon dont de multiples UT sont disposées dans un LCU, et la Figure 6 illustre un exemple de partitionnement d'une UC 32x32 en UP et UT[6].

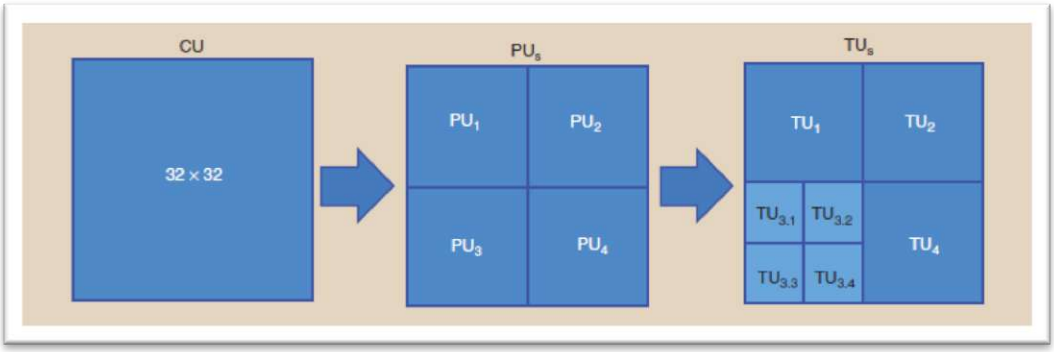


Figure 5. *Un exemple de partitionnement d'une UC 32x32 en TU etTU.*

4.2.3. L'arbre de codage

En HEVC, les unités de traitement de base dans lesquelles les images vidéo sont partitionnées peuvent être aussi grandes que 64x64 échantillons de luminance. Une application directe de la syntaxe de macrobloc H.264 /MPEG-4 AVC aux unités d'arbre de codage dans HEVC causerait quelques problèmes.

D'une part, le choix entre la prévision intra-image et la prévision compensée par le mouvement pour de grands blocs est défavorable au sens de la distorsion de la vitesse. Dans les tranches P et B, typiquement, la plupart des échantillons peuvent être bien prédits en utilisant une prédiction de mouvement compensée. Seulement pour une petite quantité d'échantillons, le codage intra-image est avantageux dans le sens de la distorsion de débit [7].

Si la norme permettrait de choisir entre la prédiction compensée par le mouvement et le codage intra-image uniquement au niveau des unités d'arbre de codage, il en résulterait une perte significative de l'efficacité de codage. En fait, il a été montré que l'activation de la décision entre le codage intra-image et inter-image à des unités inférieures à un 16x16 macrobloc peut augmenter l'efficacité de codage. D'autre part, pour permettre de telles structures de blocs fines pour la prédiction de mouvement compensé comme dans

H.264/MPEG-4 AVC, le concept de macroblocs et de modes de sous-macrobloc devrait être étendu sur des niveaux de hiérarchie supplémentaires Donnant une syntaxe compliquée [7].

De plus, si la taille de la transformation ne peut pas changer dans un bloc d'arbres de codage, l'encodeur ne peut pas s'adapter aux statistiques locales d'une image vidéo.

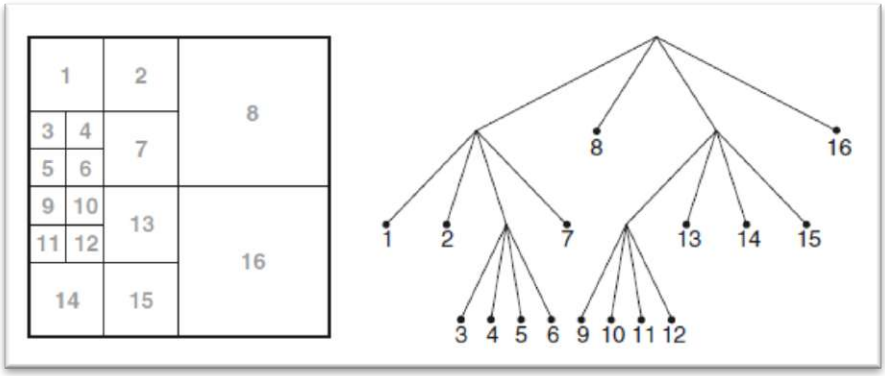


Figure 6. *Exemple pour le partitionnement d'une unité d'arbre de codage 64x64 (CTU) dans des unités de codage (UC) de 8x8 à 32x32 échantillons de luma*

Le partitionnement peut être décrit par un quad-tree, également appelé arbre de codage, qui est montré à droite. Les numéros indiquent l'ordre de codage des UC, ce qui donne une syntaxe compliquée. De plus, si la taille de la transformation ne peut pas changer dans un bloc d'arbres de codage, l'encodeur ne peut pas s'adapter aux statistiques locales d'une image vidéo[7].

4.3.La transformation dans la norme HEVC

La phase de transformation réduit les corrélations restantes du bloc résiduel, calculé comme la différence entre l'original et les blocs prédits. Le but de la transformation est de concentrer le signal résiduel en autant de coefficients que possible dans le domaine de la transformation. Dans le domaine spatial, le signal résiduel est réparti entre les échantillons des blocs, tandis que l'objectif de transformation des domaines consiste à se concentrer autant que possible le signal résiduel en quelques coefficients de domaine de transformation présentant une large amplitude et le reste peut être considéré comme négligeable. Cette idée du compactage énergétique est la propriété principale de la phase de transformation. La plupart des transformées utilisées dans les systèmes de codage vidéo normalisés appartiennent à la famille de la Transformée Trigonométrique Discrète (DTT). Parmi ceux-ci, la transformée de cosinus discrète (Discrete Cosine Transformou DCT) du type II a reçu une attention considérable dans le passé et est la transformée standard de facto utilisée dans les codecs de l'UIT et MPEG depuis MPEG-1 / H.261. Des choix supplémentaires ont été introduits

récemment, en particulier dans HEVC, où la Transformée Sinus Discrete (Discrete Sinus Transform ou DST) du type VII a été adoptée[8].

4.4.La quantification

La quantification, appliquée dans le domaine de transformation, est utilisée comme opérateur d'approximation, ce qui réduit la quantité de valeurs de sortie possibles. Dans des normes comme HEVC, la quantification est scalaire: chaque coefficient est approximativement indépendant de ses valeurs voisines. Dans ces schémas de codage, l'étape de quantification est contrôlée par un paramètre de quantification (QP) qui conteste tout coefficient dont le niveau d'énergie est inférieur à un certain seuil. Les coefficients d'énergie élevés sont également affectés par la quantification. Il convient de noter que la quantification est la seule étape non réversible dans l'ensemble du système de codage vidéo hybride, ce qui induit un codage vidéo à perte. Le codage vidéo Sans perte (ou presque sans perte)peut être atteint en n'utilisant pas de quantification dans le processus[8].

4.5.La Prédiction Intra

Le codage intra-trame, parfois appelée prédiction spatiale, est utilisée dans le codage vidéo pour éliminer les redondances spatiales en éliminant la corrélation au sein des régions locales d'une image, Le principe de base de la prédiction intra est basé sur le fait que la texture d'une région image est similaire à la texture de son voisinage et peut être prédite à partir de là. Les images codées à l'aide de cette technique sont exclusivement nommées I-slices.

Différents modèles de prédictions peuvent être utilisés à l'aide de projections de blocs décodés adjacents. Ces modèles comprennent des projections directionnelles, des projections de gradient (appelées Planar) et la projection de la valeur moyenne (dite DC).

Les Modes de Prévision Intra (IPM) sont utilisés pour dériver les prédictions du bloc courant à partir de ses limites disponibles, formées par des blocs reconstruits[8].

Le codage intra à l'intérieur de HEVC, En ce qui concerne les normes précédentes, telles que H.264 / MPEG-4 AVC, qui a 9 modes de prédiction différents pour le codage intra, HEVC a un système de prévision amélioré, avec 35 direction des modes de prédiction différents. Le mode de prédiction intra 0 se réfère à la prévision intra planaire, le mode 1 se réfère à la prédiction de DC, et le mode 2 à 34 se réfère à des modes de prédiction angulaire avec une directivité différente. La partie supérieure gauche de la Figure 7 illustre ça.

En fonction de l'IPM sélectionnée, les résidus présentent des motifs différents, de même que leurs coefficients transformés. La partie supérieure droite de la Figure I.8présente la moyenne des résidus HEVC 4×4 par mode de balayage, ainsi que leur représentation moyenne dans le domaine de la transformation par le DST-VII. Les profils résiduels moyens ont des valeurs plus basses (foncées) près des frontières disponibles, qui augmentent avec la distance des limites: les résidus issus des IPM horizontaux et verticaux n'ont que les limites gauche et supérieure respectivement, tandis que les IPM restants ont tendance à avoir à la fois Frontières disponibles[8].

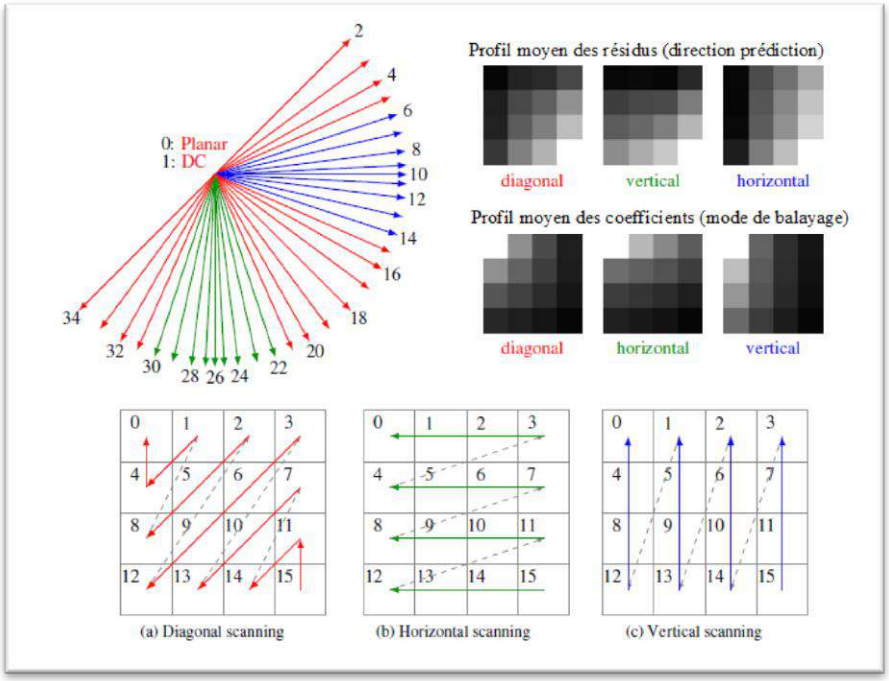


Figure 7. *Les modes d'intra-prédiction HEVC par rapport à leurs modèles de balayage avec quelques exemples résiduels moyens dans le domaine de transformation pour 4×4 UT*

On observe également que les modèles de balayage correspondent raisonnablement bien aux coefficients transformés dans chaque cas, en les triant dans un ordre croissant.

Ces motifs dans le domaine de la transformation déterminent des ordres de balayage différents (en différentes couleurs), comme présenté dans la partie inférieure de la Figure I.8. Un mode adapté à chaque motif assure, en moyenne, un ordre correct des coefficients qui regrouperont ensemble toutes les valeurs nulles. Les motifs sont décrits pour les blocs de 4×4, et le même modèle est utilisé sur les tailles de bloc plus élevées, qui sont récursivement divisé en 4 sous-blocs jusqu'à ce que la taille 4×4[8].

La Figure 8 Représente la différence entre les résidus dérivés d'un mode planaire et ceux dérivés d'un mode directionnel.

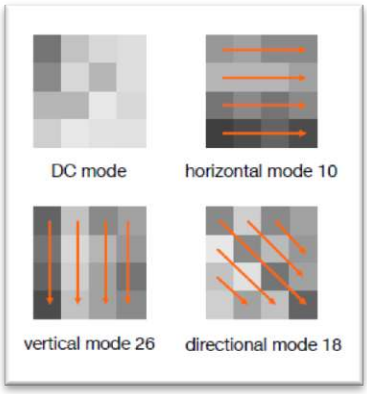


Figure 8. *Les exemples de résidus de intra-prédiction de 4x4 blocs TU extraits de séquence BQSquare en résolution de 416x240*

4.6.La Prédiction Inter

Une image ou une trame appartenant à un flux qui a été encodé à partir d'un algorithme de prédiction inter-trame. L'objectif de cet algorithme est de définir un vecteur de mouvement qui traduit le déplacement d'un bloc dans une image déjà encodée dite de référence et sa position dans l'image courante. Cette prédiction est aussi appelée prédiction temporelle. Elle a pour but de profiter des redondances temporelles entre les images ou trames voisines et ainsi obtenir un meilleur taux de compression.

La Figure 9 est une schématisation d'une séquence temporelle d'images dans le repère (x, y, t).Pour être encodée, une image est divisée en macroblocs (blocs de pixels). Le codage en inter-prédiction consiste à trouver un bloc similaire au bloc courant sur une image de référence. Ce processus est réalisé par un algorithme de Block-matching. Si l'encodeur réussit à trouver un tel bloc, le vecteur reliant les deux blocs que l'on appelle vecteur de mouvement sera enregistré et la différence entre ce vecteur et le vecteur prédit sera encodée. Ce vecteur prédit est défini par la norme et est fonction des vecteurs des macroblocs voisins déjà encodés. Le processus calcule ensuite l'erreur de prédiction c'est-à-dire la différence de valeur des composantes de luminance et de chrominance entre le bloc prédit dans l'image de référence et le bloc d'origine, aussi appelée bloc résiduel. Ce dernier sera encodé dans le flux binaire après des étapes de transformée et de quantification. En codage vidéo, cette étape de recherche de vecteur est appelée estimation de mouvement et celle du calcul des différences est la compensation de mouvement.

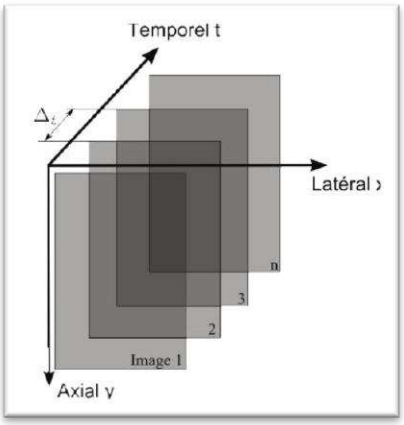


Figure 9. *Représentation schématique d'une séquence temporelle d'images*

Les algorithmes de compression vidéo tels que HEVC utilisent la prédiction inter-images pour réduire les données vidéo entre une série d'images. Cela implique des techniques telles que le codage différentiel, où une image est comparée à une image de référence et seuls les pixels qui ont changé par rapport à cette image de référence sont codés. Cela permet de réduire le nombre de valeurs de pixels codées et envoyées. Lorsqu'une séquence encodée de cette façon est affichée, les images apparaissent comme dans la séquence vidéo d'origine.

La Figure 10: Avec le codage différentiel, seule la première image (image I) est codée dans son intégralité. Dans les deux images suivantes (images P), il est fait référence à la première image pour les éléments statiques (tels que la maison). Seuls les éléments en mouvement (l'homme qui court) sont codés à l'aide de vecteurs de mouvement, ce qui réduit la quantité d'informations transmises et stockées.

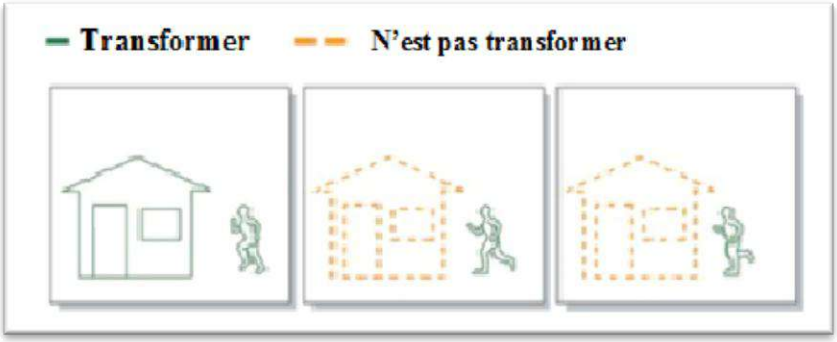


Figure 10. *Avec la prédiction inter-images, chaque image d'une séquence est classifiée comme un certain type d'image (image I, image P ou image B)*

D'autres techniques telles que la compensation de mouvement basée sur les blocs peuvent être appliquées afin de réduire encore davantage la quantité de données. Cette technique tient compte du fait qu'une bonne partie de ce qui compose une nouvelle image dans une séquence vidéo se retrouve dans une image précédente, bien qu'à un emplacement potentiellement différent. Elle divise une image en série de macro-blocs (blocs de pixels). Il est possible de composer ou de « prédire » une nouvelle image bloc par bloc en recherchant

un bloc identique dans une image de référence. Si un bloc référent est trouvé, l'encodeur code l'emplacement dans l'image de référence où se situe le bloc identique. Le codage du vecteur de mouvement utilise moins de bits que le codage du contenu réel d'un bloc.

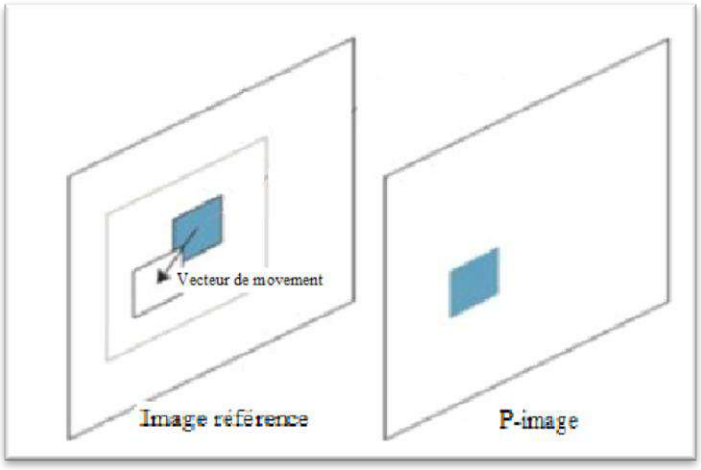


Figure 11. *Illustration de la compensation de mouvement basée sur bloc*

Une image I, ou intra-image, est une image autonome qui peut être décodée indépendamment, sans référence à d'autres images. La première image d'une séquence vidéo est toujours une image I. Les images I sont nécessaires en tant que points de départ pour de nouveaux téléspectateurs ou en tant que points de resynchronisation si le flux binaire transmis est endommagé. Les images I peuvent être utilisées pour l'implémentation de fonctions d'avance rapide et de rembobinage et autres fonctions d'accès aléatoire. Un encodeur insère automatiquement des images I à intervalles réguliers ou sur demande s'il est prévu que de nouveaux clients rejoignent les autres dans la visualisation d'un flux. L'inconvénient des images I est qu'elles consomment beaucoup plus de bits mais, d'un autre côté, elles ne génèrent pas beaucoup d'artéfacts, qui sont dus aux données manquantes.

Une image P est une inter-image prédictive qui fait référence aux parties des images I et/ou P antérieures pour le codage de l'image. Les images P nécessitent généralement moins de bits que les images I, mais elles sont généralement très sensibles aux erreurs de transmission en raison de la dépendance complexe vis-à-vis des images P et/ou I antérieures.

Une image B est une inter-image bi prédictive qui fait référence à la fois à une image antérieure et à une image future. L'utilisation d'images B entraîne une augmentation de la latence.

Séquence type avec des images I, B et P. Une image P peut uniquement faire référence aux images I ou P précédentes, alors qu'une image B peut faire référence aux images I ou P précédentes et suivantes[5].

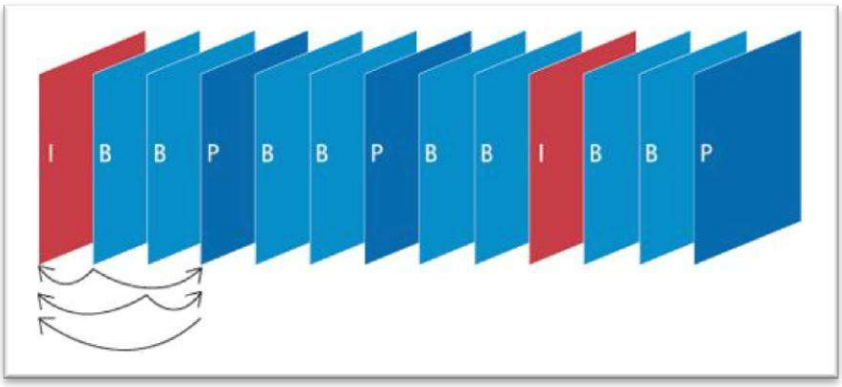


Figure 12. *Types de prédictions dans un ensemble d'images (Groupe of Pictures)*

Usuellement, une séquence vidéo est segmentée selon des groupes d'images consécutifs (GOP : Group of Pictures) débutant par une image l et suivis d'images de type P puis B dans des positions bien spécifiées (exemple d'ordre d'affichage: (I P P I P …..). Les images constituant le GOP peuvent avoir des ordonnancements différents suivant l'encodage ou l'affichage.

Nous considérons 3 tailles de GOP:

GOP3 : (I P P) ;

GOP7 : (I B B P B B P I) ;

GOP12 : (I B B P B B P B B P B B I),

4.7.Le Filtrage adaptatif

HEVC utilise deux filtres offset d'adaptation d'échantillon (SAO) et le filtrage effectué dans la boucle de codage (In-Loop Filters) qui tentent de réduire les artefacts de compression et d'améliorer l'efficacité du codage. Dans l' HEVC Le filtre de la boucle de codage est traité les bords sur une grille de 8 pixels et Une complexité informatique plus faible que H.264 / AVC qui utilise Une grille de 4 pixels[9].

SAO consiste à sélectionner un type de décalage pour chaque pixel en fonction de son Pixels voisins et en ajoutant le décalage. La boucle de codage et SAO peuvent être implémentés

dans un seul étage de pipeline. Cela permet de planifier l'ordonnancement dans le filtre de déblocage Indépendamment de la structure arborescente codante. Une granularité plus petite peut également être utilisée pour Sauvegarder la mémoire tampon de pipeline SRAM au prix de la complexité d'ordonnancement. Comme le processus de filtrage en boucle pour le bloc de pixels actuel dépend des blocs à droite et à gauche qui n'ont pas encore été reconstruits, le bloc entier ne peut pas être traité complètement. La sortie du filtre de déblocage est décalée de

l'entrée par quatre pixels de luma et deux pixels de chrominance vers la gauche et vers le haut, et la sortie de SAO est décalée d'un autre pixel pour toutes les composantes de couleur dans les deux directions[9].

Sample Adaptive Offset (SAO) :

Dans u filtre SAO chaque pixel est classé en une des quatre bandes ou un des quatre types de bordures et ajouté Un décalage. Pour les décalages de bandes, la bande de chaque pixel dépend de sa valeur et La position des quatre bandes. Pour les décalages de bord, le bord de chaque pixel dépend de Si sa valeur est plus grande ou plus petite que deux de ses voisins[9].

La sélection entre les décalages de bande et les décalages de bord, la position des bandes, le choix des voisins pour les décalages de bord et les valeurs des décalages sont signalés au niveau CTU pour le luma et la chroma séparément. SAO a des dépendances sur les pixels voisins similaires à la prédiction intra et Par conséquent, une gestion de flux de données similaire doit être utilisée.

Comme dans la prédiction intra, un tampon de rangée supérieure de taille de largeur d'image et un tampon de colonne gauche de taille CTU-taille sont nécessaires. Ces tampons stockent les pixels pré-SAO et leurs paramètres SAO. Cependant, Contrairement à la prévision intra, le choix de la granularité des pipelines est très souple et peut être Choisi en fonction des exigences de débit. Contrairement au filtre de boucle qui fonctionne Sur une base de bord, SAO fonctionne sur une base par pixel. Ainsi, les deux filtres en boucle ont comparé la complexité de calcul, même si le calcul des SAO qui implique la comparaison principale et l'addition[9].

4.8.Le Codage entropique/ CABAC

Le codage entropique est appliqué pour coder tous les éléments de syntaxe et les coefficients de transformée quantifiés. Dans H.264 / AVC, le codage de longueur variable adaptatif au contexte (CAVLC) est le codeur d'entropie de base, et le codage arithmétique binaire adaptatif au contexte (CABAC) est facultativement utilisé dans les profils principal et élevé. CABAC peut fournir une meilleure efficacité de codage que CAVLC en raison de son moteur de codage arithmétique et de modélisation de contexte plus sophistiquée. Bien que CABAC améliore l'efficacité du codage, il augmente la complexité du codage. Ceci est plus prononcé à des débits binaires plus élevés (petits paramètres de quantification (QPs)), où les données de coefficients de transformation ont un rôle dominant dans les flux binaires codés[6].

En HEVC, pour améliorer le pire des cas, le codec utilise un mode alternatif à débit plus élevé pour coder les données de coefficient de transformée. La Figure 13 illustre le schéma de blocage du codage d'entropie HEVC[6].

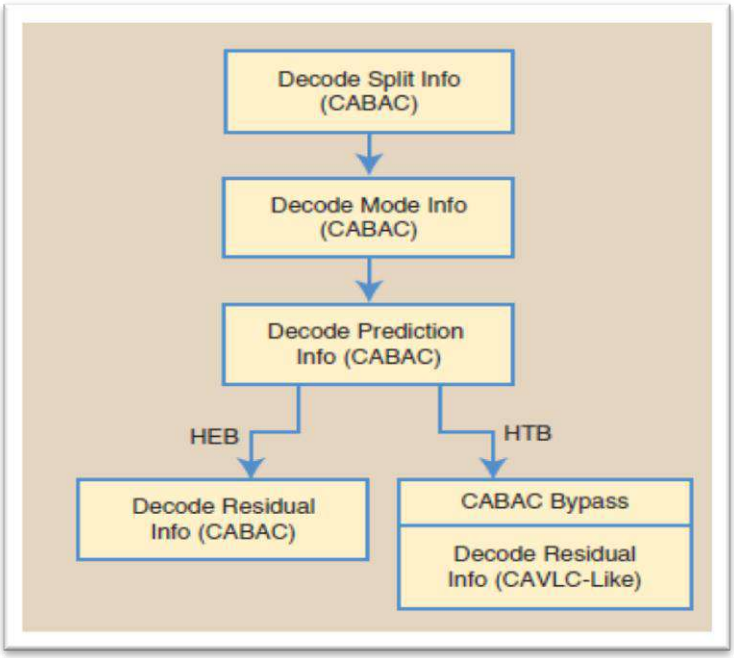


Figure 13. *Codage entropique dans HEVC*

Le codage CABAC a montré sa capacité à améliorer l'efficacité de la compression de 10à 15% par rapport au codage CAVLC mais aux dépends d'un traitement beaucoup plus complexe. Le codage arithmétique permet d'affecter un nombre non entier de bits aux symboles. Le codage CABAC atteint les meilleurs taux de compression par:

a. la sélection de modèles de probabilité pour chaque élément syntaxique selon son contexte,
b. l'adaptation des estimations de probabilités basées sur les statistiques locales, et
c. l'utilisation du codage arithmétique à la place du codage à longueur variable. Le codage d'un symbole passe par les étapes suivantes[6]:
    1. "Binarisation": CABAC utilise le codage arithmétique binaire où uniquement des décisions binaires (0 ou 1) sont codées. Tout symbole de valeur non binaire tel qu'un coefficient transformé ou un vecteur de mouvement, doit alors être converti en un code binaire avant le codage arithmétique, c'est la binarisation. Les étapes suivantes (2, 3 et 4) sont répétées pour chaque bit (ou « bin ») du symbole binarisé:
    2. Sélection du modèle du contexte. Un modèle de contexte est un modèle de probabilité pour un ou plusieurs bins du symbole binarisé et est choisi parmi une sélection de modèles disponibles en fonction des statistiques des symboles

de données récemment codés. Le modèle contextuel stocke les probabilités de la valeur (0 ou 1) de chaque bin.

3. Codage arithmétique: le codeur arithmétique code chaque bin selon le modèle de probabilité sélectionné. Il n'existe que deux sous-ensembles pour chaque bin correspondant à 0 ou à 1.
4. Mise à jour des probabilités: le modèle contextuel sélectionné est mis à jour en fonction de la valeur codée actuelle. Par exemple, si la valeur de bin est 1, le compteur des fréquences de 1 augmente.

5. Les formats de vidéo

Idéalement, la couleur de chaque pixel d'une image numérique doit être décrite par la quantité de rouge, de vert et de bleu qui la composent.

Les couleurs sont échantillonnées au format 4:4:4, ce qui signifie que les trois couleurs primaires (rouge, vert, bleu) sont systématiquement quantifiées pour chaque pixel. On peut l'illustrer de la manière suivante sur cette section d'image numérique de 4 pixels sur 4 [10]:

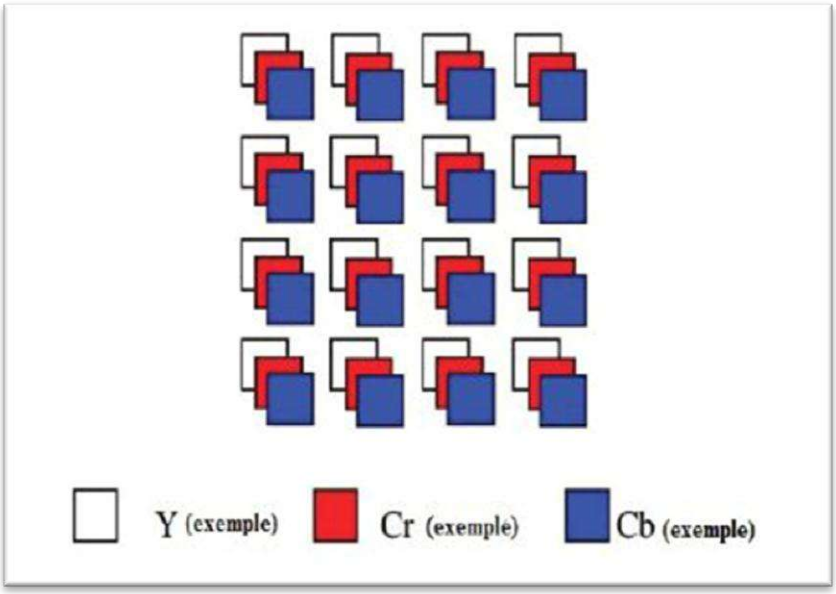


Figure 14. *Exemple de format 4 :4 :4*

Avec les autres formats d'échantillonnage (4:2:2; 4:2:1; 4:2:0,...), la description des couleurs des images numériques n'est pas aussi précise mais les fichiers d'images sont moins lourds : il y a deux fois plus d'informations dans une image 4:4:4 que dans une image 4:2:2.

En dessous du format 4:4:4, la couleur des pixels n'est plus décrite par ses composantes rouge, vert et bleu (R, V, B) mais par sa luminance (Y) et deux mesures de chrominance : Cb pour le bleu et Cr pour le rouge. Les coordonnées R, V, B de chaque pixel

deviennent donc : Y, Cb, Cr. Cette technique exploite les limites de la vision humaine plus sensible à la luminosité des pixels qu'à leur couleur.

Le format d'échantillonnage 4 :2 :2 accorde deux fois plus d'importance aux informations relatives à la luminance qu'à la chrominance : alors que la luminance (Y) est mesurée de manière systématique pour chaque pixel, les informations relatives aux couleurs ne sont données que pour un pixel sur deux. Ce format d'échantillonnage permet d'obtenir une qualité d'image très satisfaisante tout en réduisant sensiblement la taille des fichiers de vidéo par rapport au 4:4:4. Le 4:2:2 et tous les formats d'échantillonnage inférieurs constituent à ce titre une forme de compression des images numériques.

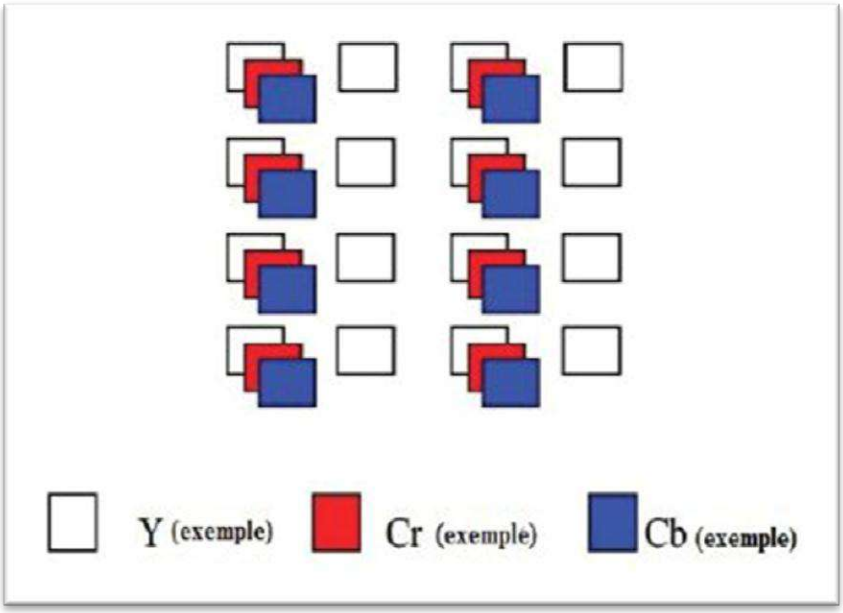


Figure 15. *Exemple de format 4 :2 :2*

Les autres formes d'échantillonnage se traduisent par une compression plus importante des informations relatives à la couleur des pixels. Avec le format 4 :2 :0 par exemple, la luminance de chaque pixel est systématiquement décrite mais les informations sur la chrominance ne concernent qu'un pixel sur deux et seulement sur une ligne d'image sur deux. Le format 4:2:0 est notamment utilisé pour la télévision haute définition (TVHD).

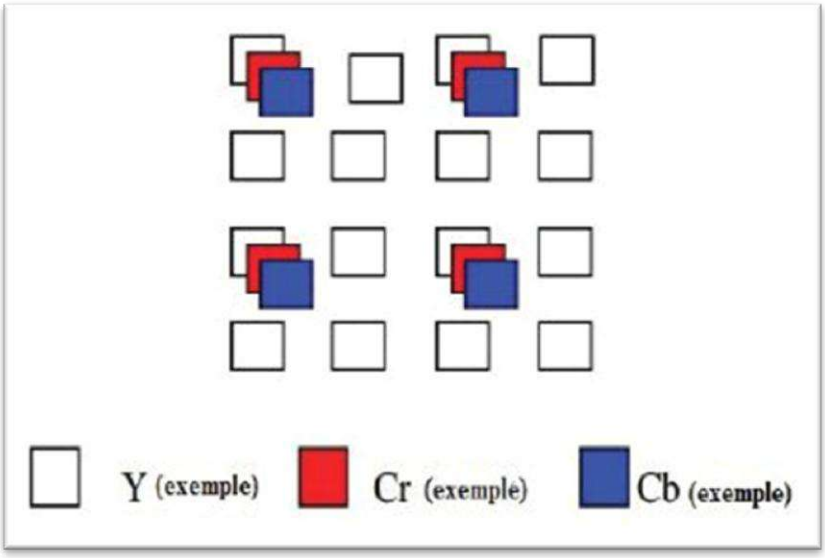


Figure 16. *Exemple de format 4 :2 :0*

6. Les niveaux de parallélisme pour l'encodeur HEVC

Afin de surmonter les limites des stratégies de parallélisation employées en H.264/ MPEG-4 AVC, HEVC fournit des outils de codage basés sur VCL qui sont spécifiquement conçus pour permettre le traitement sur des architectures parallèles de haut niveau[11].

Trois nouveaux outils visant à faciliter le traitement parallèle de haut niveau ont été la norme HEVC.

6.1.Slice

La segmentation de haut niveau d'une image en HEVC est similaire à celle de H.264 / MPEG4 AVC basée sur le concept de tranche (slice).

HEVC introduit la nouvelle fonctionnalité de la fragmentation slice au niveau conceptuel deux niveaux distincts.

- Pour le premier niveau de fragmentation, chaque slice peut être divisée en un ou plusieurs slice de segments aux limites de CTU. Le premier segment de slice de segment (dans l'ordre de balayage de trame CTU) est slice de segment indépendant et inclut l'en-tête de slice complète (segment). Slice de segment indépendantes est également souvent appelé une slice régulière, car il est conceptuellement équivalent à ce qui est spécifié dans H.264/MPEG-4 AVC.
- Le deuxième niveau de fragmentation de HEVC est donné par l'instrument des sous-ensembles slices de segments, souvent appelés sous-flux. Chaque sous-ensemble de slice de segment contient tous les bits codés d'un sous-ensemble de CTU couvert par la tranche correspondante Des données de segment et un décalage de point d'entrée

est fourni pour chaque sous-ensemble (mais le premier) dans l'en-tête de segment de slice correspondant pour une identification de sous-ensemble correcte.

Slice Segments :

La fragmentation des slices par l'utilisation de slice de segment, une slice en HEVC est définie comme un ensemble slice de segments, où le premier segment d'une slice est le slice de segment indépendant[12], comme illustré à titre d'exemple dans la Figure 17.

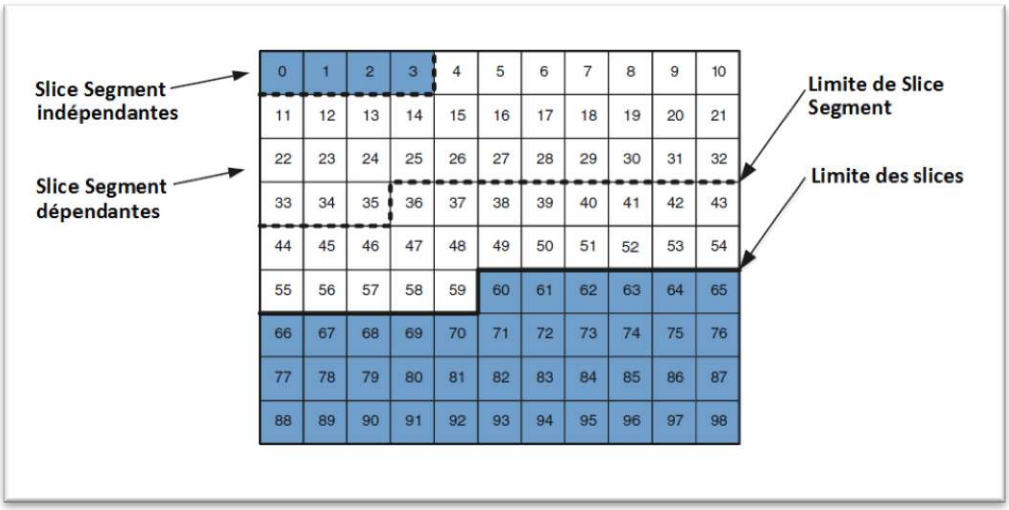


Figure 17. *Structure de l'image en HEVC à l'aide de slices, montrant les CTU appartenant à des segments indépendants de slices (bleu ombré) et slice des segments dépendants de deux tranches d'une image*

Les slices des segments dépendants ne contiennent qu'un ensemble minimum de paramètres d'en-têtes de slice et son utilisation est indiquée par l'élément de syntaxe d'en-tête de slice *dependent_slice_segment_flag*.

Bien que toutes les données de segments de tranches dépendantes soient transportées par son propre train de bits CABAC, par conséquent, le moteur CABAC doit être purgé et réinitialisé aux limites de segment, Le contenu adapté de toutes les variables de contexte CABAC est mémorisé à la fin de chaque slice de segment afin d'être réutilisé pour l'initialisation dans le segment de tranche dépendante suivant.

Notez que pour chaque CTU dans un slice de segment, l'élément de syntaxe *end_of_slice_segment_flag* est signalé pour indiquer si la CTU correspondante est la dernière CTU dans le slice de segment.Un cas d'utilisation des segments de tranches dépendantes est la réduction du retard de bout en bout pour des applications à retard ultra-faible, telles que la contribution à distance de vidéo ou de diffusion. Comme l'en-tête de slice de segment dépendante ne porte qu'un minimum de données, des segments de tranche peuvent également être utilisés comme points d'entrée pour des techniques de parallélisation.

6.2.Tile

Un mécanisme de partitionnement d'images similaire à des slices, qui est basé sur une subdivision flexible de l'image en régions rectangulaires de CTU, de sorte que des dépendances de codage entre des CTU de différentes partitions sont interdites.

Lorsque les Tiles sont activés dans HEVC, l'image est divisée en groupes de CTU rectangulaires séparés par des frontières verticales et /ou horizontales[11]. L'élément de syntaxe *PPStiles_enabled_flag*indique l'utilisation de Tile. L'approche de la Tile présente certaines similitudes avec les groupes de slice en H.264 / MPEG-4 AVC en utilisant la fonction d'ordonnancement de macroblocs flexible(flexible macrobloc kordering) (FMO) avec la carte de groupe de slice de type 2.

Le nombre de Tiles et l'emplacement de leurs limites peuvent être définis pour la séquence entière ou changés d'image en image. Ceci est obtenu dans le PPS par des paramètres de signalisation tels *quenum_tile_columns_minus1*, *num_tile_rows_minus1*, *uniform_spacing_flag*, *column_width_minus1* and *row_height_minus1*.

Avec ces éléments de syntaxe, le nombre de Tiles, ainsi que le cloisonnement des images dans les Tiles peuvent être définis. Puisque la signalisation des Tiles est incluse dans le PPS, Cette structure peut changer sur une base d'image. Ceci a pour avantage qu'un codeur peut utiliser différentes manières d'agencer les Tiles dans différentes images afin de contrôler l'équilibrage de charge entre les noyaux utilisés pour le codage / décodage.

Le fait qu'une tuile puisse être traitée indépendamment des autres tuiles d'une image rend les carreaux aussi très appropriés pour les environnements de transmission avec perte, si les carreaux d'une image sont transportés dans des paquets différents.

Les Tiles modifient l'ordre de balayage régulier des CTU à partir d'un ordre de balayage de trame à base d'image (c'est-à-dire, les CTU sont traitées en rangée de gauche à droite) à un ordre de balayage de trame à base de Tile, dont un exemple est représenté sur la Figure 18.

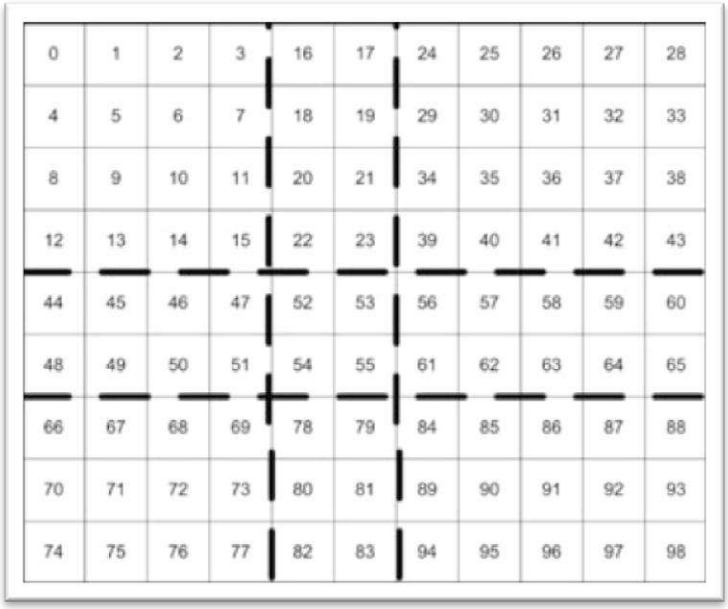


Figure 18. *Ordre de balayage de trame basé sur Tiles de CTU avec neuf Tiles de tailles différentes dans l'image. Notez que les limites des Tiles sont indiquées par des tirets gras.*

Etant donné que l'ordre de balayage proprement dit peut être différent de l'ordre de balayage de trame d'image utilisé dans des systèmes de traitement d'image typiques, les décodeurs peuvent traiter les CTU de Tiles toujours dans un mode de balayage de trame basé sur image n'utilisant aucune parallélisation. Notez que cela peut être obtenu en stockant et en rechargeant les valeurs de toutes les variables de contexte CABAC pour chaque Tile aux limites de la Tile tout en passant d'une ligne CTU à la suivante.

Pour chaque slice segment et Tile, au moins une des conditions suivantes doit être vraie dans une image: Toutes les CTU dans un slice segment appartiennent à la même Tile ou toutes les CTU d'une Tile appartiennent au même slice segment. Un exemple de cette situation est donné à la Figure 19, qui montre le même partitionnement d'une image en deux Tiles avec deux modes de fragmentation différents en slice et slice segment [7].

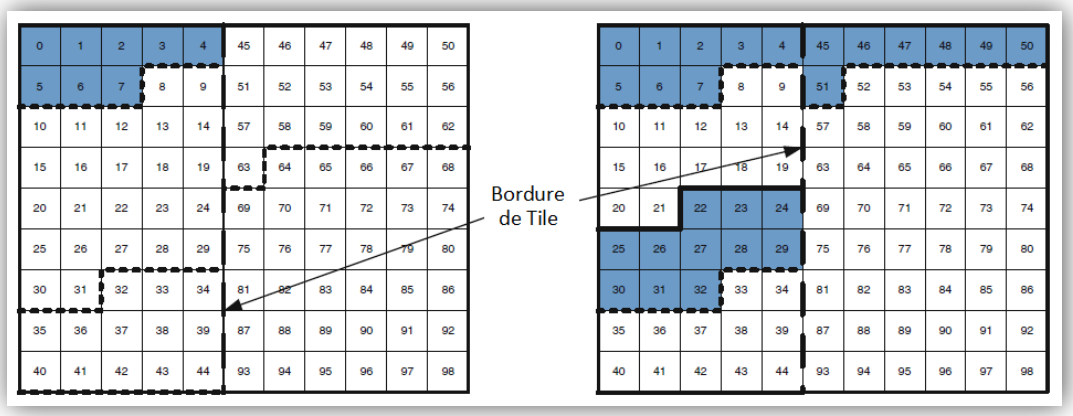


Figure 19. *Exemple de partitionnement de Tiles en utilisant deux Tiles en slice et slice de segments.*

La Figure 19 présente deux façons différentes de fragmenter un exemple de partitionnement de Tiles en utilisant deux Tiles en slice et slice de segments. Gauche: Une slice incluant quatre slice de segments dépendants. Droite: Trois slice, chacune contenant un slice de segments dépendante. Notez que les limites des Tiles sont marquées avec des lignes pointillées en gras, les limites de la tranche avec des lignes pleines en gras et les limites entre les slice de segments avec des lignes pointillées. Les CTU appartenant aux slice indépendantes de slice sont bleue.

6.3.Wpp(WavefrontParallelProcessing)

Une approche de traitement parallèle le long du principe d'ordonnancement de front d'onde, qui est basé sur un partitionnement de l'image en lignes CTU de telle sorte que les dépendances entre CTU de différentes partitions, tant en termes de codage prédictif et codage entropique, sont préservées dans une large mesure[7].

La Figure 20, en mode WPP, chaque ligne CTU est traitée par rapport à sa ligne CTU précédente en utilisant un retard de deux CTU consécutives. De cette façon, aucune dépendance entre des lignes CTU consécutives n'est pas rompues aux limites de la partition à l'exception des variables de contexte CABAC à la fin de chaque ligne CTU.

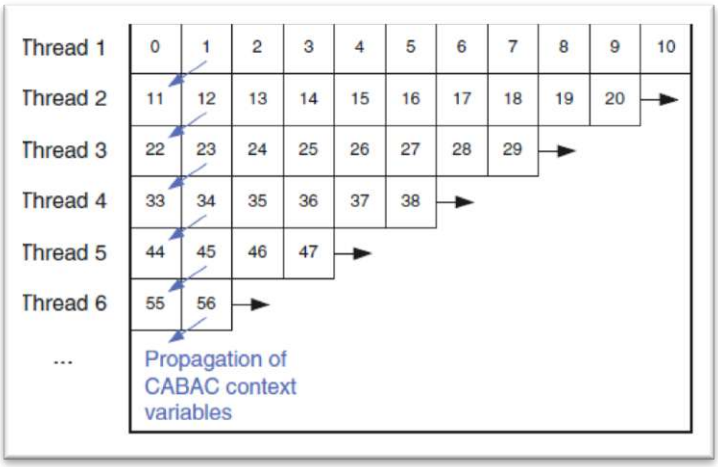


Figure 20. *Illustration de WPP avec des threads*

Pour atténuer la perte potentielle d'efficacité de codage qui résulterait de l'initialisation CABAC classique au point de départ de chaque ligne CTU, le contenu des variables de contexte CABAC (partiellement) adaptées est propagé à partir de la seconde CTU codée / décodée de la ligne CTU précédente À la première CTU de la rangée actuelle CTU, comme montré à la Figure 20.

En conséquence, les pertes d'efficacité de codage introduites par WPP sont relativement faibles par rapport au cas d'un codage d'image n'utilisant pas WPP mais avec des réglages par ailleurs identiques. En outre, WPP ne modifie pas l'ordre de balayage raster

ordinaire des CTU. En outre, en utilisant une technique de transcodage CABAC relativement simple, un flux binaire WPP peut être transcodé vers ou à partir d'un flux binaire non WPP sans aucune modification du processus de reconstruction d'image.

Lorsque WPP est activé, un nombre de threads jusqu'à le nombre de lignes CTU dans une image peut travailler en parallèle pour traiter les différentes lignes CTU,

En utilisant le traitement WPP dans un décodeur, chaque fil de décodage traite une seule ligne CTU de l'image.

En tant que surcharge additionnelle supplémentaire dans le traitement, WPP requiert de mémoriser le contenu de toutes les variables de contexte CABAC après avoir terminé le codage/décodage de la seconde CTU dans chaque ligne CTU. Par ailleurs, WPP n'exige pas de manipulation supplémentaire des bordures de partition pour préserver les dépendances utilisées par le codage/décodage d'entropie.

Une analyse en profondeur des outils de parallélisation inclus dans HEVC a montré que lorsque WPP est combiné avec l'algorithme OWF(overlapped wave front, plusieurs images peuvent être décodées simultanément), il offre une meilleure Évolutivité de la parallélisation que les tuiles.

6.4.Comparions entre les outils de traitement parallèle

Les Tiles sont avantageux pour le décodage en parallèle, par rapport aux Slices ou aux WPP. Dans l'exemple illustré à la Figure 21, une image est divisée en 4 sous-images de taille égale[13].

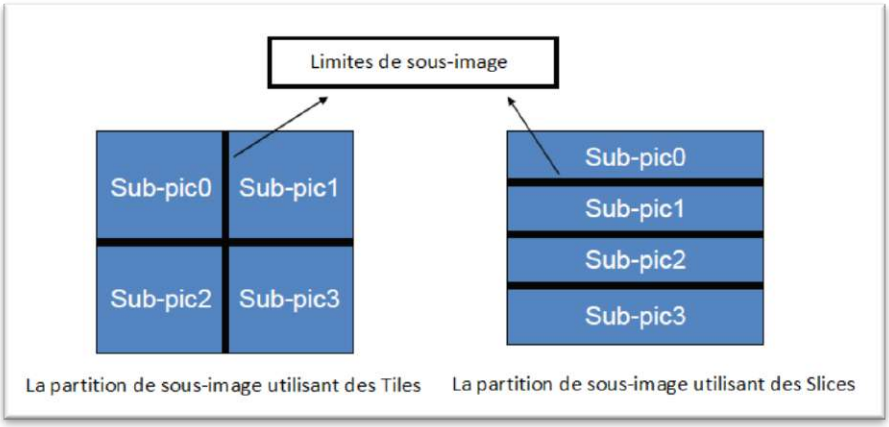


Figure 21. *Partitionnement en sous-images*

Avec les Tiles, l'image peut être divisée en sous-images 2x2. Cependant, avec des slices régulières, par exemple, l'image devrait être divisée en 1x4 sous-images. Le cloisonnement de sous-image à l'aide de Tiles est apparemment avantageux dans les aspects suivants:

- la largeur de la sous-image est seulement la moitié de la largeur de l'image, ce qui coupe la taille du tampon de ligne de moitié.
- moins de limites de sous-images en termes de pixels (une frontière verticale et horizontale par rapport à trois limites horizontales de sous-image comme le montre la Figure 21) conduit à une meilleure qualité; Par exemple, pour 1080 HD (1920x1088), les tuiles se traduiraient par des frontières de 3008 pixels par rapport aux limites de 5760 pixels pour les slices
- les tiles sont indépendantes, ce qui minimise les communications entre cœurs; Seules les données d'image de référence sont partagées entre les noyaux.
- peuvent être simplement répliqués pour construire des solutions multi-core sans avoir besoin d'efforts supplémentaires de validation de conception pour vérifier les interconnexions croisées.

Pour faire travailler les Tiles pour le traitement de décodage parallèle sur la plate-forme multi-core, plusieurs restrictions devraient être imposées sur le flux binaire afin qu'un décodeur puisse compter sur des fonctionnalités de traitement parallèle intégrées pour envoyer simultanément le flux binaire vers le multi-noyau pour le décodage parallèle. Ces restrictions sont les suivantes:

- Il devrait être mandaté de diviser une image Ultra-HD en un certain nombre de carreaux uniformément espacés de sorte que l'équilibrage de taux de pixel est garanti pour les plates-formes de noyau multiples. La largeur des Tiles ne doit pas dépasser le partitionnement de la largeur de la mémoire tampon de ligne unique, de sorte que la mémoire tampon de ligne du décodeur mono cœur n'a pas besoin d'être augmentée lorsque des noyaux simples sont répliqués pour constituer un décodeur multi-core.
- Les Tiles doivent être indépendants pour minimiser la communication de données entre les noyaux. Seul le filtre de déblocage, le filtre d'adaptation adapté (SAO), le filtre adaptatif en boucle (ALF) peuvent traverser les limites des tiles.
- Les Slices, les Slices d'entropie et les WPP, si présents, doivent être contenus dans des tiles et ne peuvent pas traverser les limites
- Les entrées de sous-flux de tiles dans le flux binaire sont signalées au niveau de l'image, de sorte qu'un décodeur multi-core peut envoyer des sous-courants vers des noyaux en parallèle.

Avec des Tiles uniformément espacés et des restrictions ci-dessus, HEVC permet potentiellement un moyen rentable de construire des plates-formes multicœurs pour soutenir Ultra-HD décodage en temps réel. Comme le montre la Figure 22, un décodeur de 8kx4k à 30 fps peut être construit en répliquant simplement le décodeur 4kx2k à 30 fps à noyau unique

quatre fois et en ajoutant un noyau de traitement de frontière de sous-image pour exécuter, par ex. Filtre anti-blocage, Offset adaptatif d'échantillons (SAO), Filtre adaptatif en boucle (ALF) le long des limites des tiles[13].

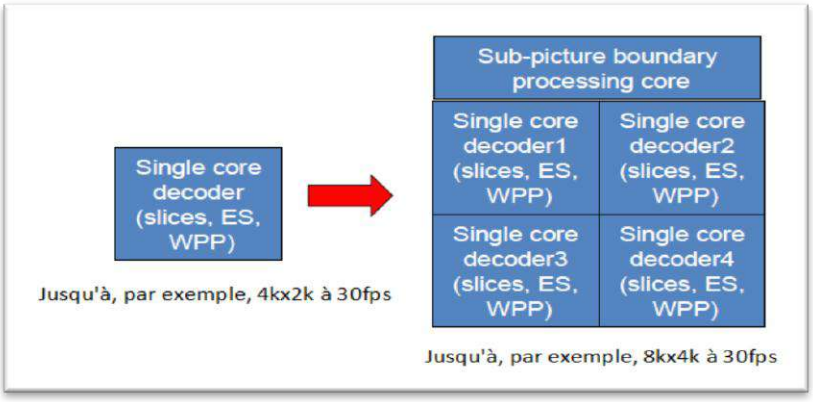


Figure 22. *Construction d'un décodeur multi-core en répliquant des décodeurs à noyau unique*

7. Transformation des couleurs

Le système de stockage colorimétrique est modifié par un système moins gourmand en taille mémoire. Par exemple, le codec MPEG (1,2 ou 4) transforme le classique système RVB (image de gauche) qui enregistre pour chaque pixel la valeur du rouge, du vert et du bleu, en un système plus proche de la représentation de l'œil humain et sollicitant moins d'espace de stockage, le système YUV (appelé aussi Y-Cr-Cb)représentent le signal de luminance (luma Y, c'est-à-dire l'information en noir et blanc) et deux signaux de chrominance (chroma Cb et Cr aussi appelé U et V, c'est-à-dire l'information sur la couleur) [5]comme le montre la Figure 23.

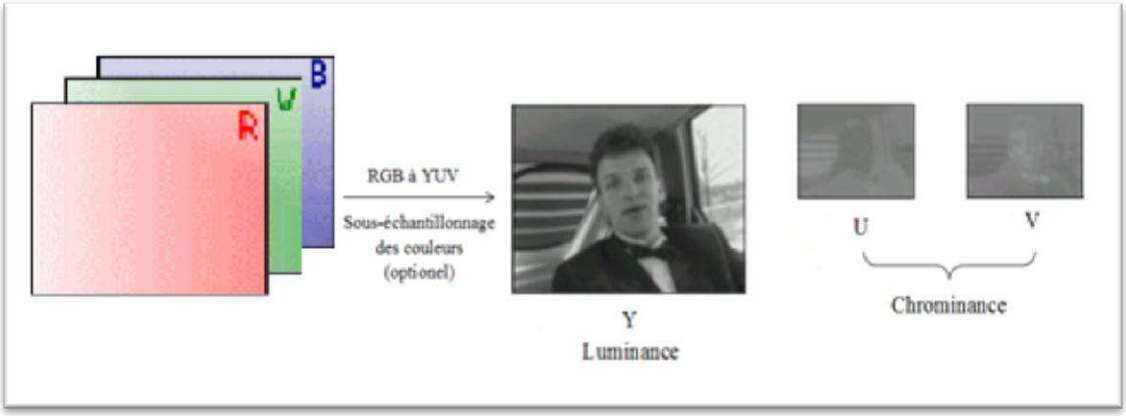


Figure 23. *Format YUV 4 :1 :1*

Voici les relations liant Y à R, G et B, U à R et à la luminance, et enfin V à B et à la luminance [22]:

$$Y = 0.299R + 0.587\,G + 0.114\,B \quad (7.1)$$

$$U = -0.147R - 0.289\,G + 0.436B = 0.492(B - Y) \quad (7.2)$$

$$V = 0.615R - 0.515G - 0.100B = 0.877(R-Y). \quad (7.3)$$

8. Conclusion

Dans ce premier chapitre, les notions de base relatives à la vidéo numérique et sa compression ont été introduites. Les normes de compression vidéo MPEGs et H.26x ont été présentées succinctement. Elles se distinguent principalement de la norme HEVC qu'elles offrent en rapport direct avec la complexité du traitement. Néanmoins, le schéma de base reste le même, basé sur le découpage en blocs, la prédiction et la compression vidéo.

Ce chapitre dans sa première partie, rappelle les principales généralités de la vidéo numérique et l'historique des différents standards et normes ainsi que les critères d'évaluations.

Dans la deuxième partie nous avons détaillé la norme HEVC ainsi que le fonctionnement et ses principales caractéristiques et la structure principale de l'encodeur HEVC. Cette partie a présenté la motivation pour la compression vidéo ainsi que quelques concepts généraux concernant une vue d'ensemble du système de codage vidéo. La transformation utilisant le partitionnement à base de blocs.

Les encodeurs vidéo actuels exploitent les redondances existantes dans les images de la séquence vidéo via des prédictions. Ces prédictions peuvent être spatiales ou temporelles (intra-prédiction et inter-prédiction), selon que la source de prédiction est la même image ou une autre image.

Une étude comparative a été présentée se basant sur Les outils de traitement parallèle (Slice/ Tile/ WPP). Nous avons trouvé que les Tiles sont avantageux pour le décodage en parallèle, par rapport aux Slices ou aux WPP.

Une étude détaillée de techniques de programmation parallèle en détail sur L'OpenMP fera l'objet du chapitre deux.

Chapitre II : Outils et techniques de programmation parallèle

1. Introduction :

Le calcul parallèle est de plus en plus utilisé comme une alternative permettant d'obtenir des diminutions substantielles du temps de restitution des calculs et/ou d'effectuer des études numériques de plus grande précision tout en gardant un temps de calcul raisonnable. Les possibilités de calcul parallèle sont aujourd'hui présentes à des degrés divers dans les systèmes informatiques. Ce peut être via des machines spécialement conçues pour ce type de calcul (CRA Y T3E, SGI Power Challenge, IBM SP), des serveurs ou stations multiprocesseurs telles que celles offertes par la plupart des constructeurs, ou bien une grappe, homogène ou non, de machines précédemment citées. Ces dernières sont de plus en plus fréquemment choisies car, construites en utilisant du matériel produit en grande série, les grappes sont plus abordables que les machines parallèles dédiées. Le problème se pose alors d'offrir au programmeur d'applications critiques les moyens d'exploiter le potentiel de puissance offert par ces architectures parallèles.

L'obtention d'un parallélisme efficace dépend fortement d'une distribution équitable de la charge de calcul entre les processeurs. La prise en compte de l'irrégularité ajoute un degré de difficulté à la programmation d'applications parallèles, notamment si l'on souhaite que le programme soit à la fois efficace et portable.

Une technique de programmation se prêtant bien à la mise en œuvre d'applications irrégulières est basée sur des réseaux de processus légers communicants. Un processus léger (ou « thread », en anglais)[14]est une abstraction permettant d'exprimer la multiprogrammation (i.e. de multiples flots d'exécution indépendants) au sein d'un processus système. Le qualificatif "léger" vient du fait que la gestion de ces entités a un coût très faible par rapport aux processus système. Il est alors intéressant d'utiliser la multiprogrammation légère pour manipuler (démarrer, arrêter, suspendre, déplacer) les tâches d'un programme dynamiquement. Cette technique est également opportune pour recouvrir les temps de communication par du calcul utile.

2. Architectures pour le calcul parallèle

Les différentes architectures matérielles se prêtant au calcul parallèle peuvent être classées selon plusieurs critères. Flynn[15] a proposé dans les années 70 une classification fondée sur la multiplicité des flots d'instructions et de données, d'où la distinction des premières machines parallèles en SIMD (machine exécutant une même instruction

simultanément sur plusieurs données) et MIMD (machine ayant plusieurs flots d'exécution simultanés sur plusieurs données). Certains auteurs [16] ont par la suite subdivisé la classe MIMD en fonction du degré de couplage entre processeurs et mémoires. On distingue ainsi les multiprocesseurs (ou machines à mémoire partagée) des multi-ordinateurs (ou machines à mémoire distribuée).Cette classification générale constitue un repère important lorsque l'on analyse l'évolution des architectures parallèles au fil des années.

Dans la suite nous tenons à différencier les architectures à mémoire partagée des architectures à mémoire distribuée, puis nous discutons des tendances actuelles qui mènent à combiner ces deux modèles de mémoire pour la construction de supercalculateurs.

2.1.Architectures à mémoire partagée

Les architectures parallèles à mémoire partagée, aussi appelées multiprocesseurs, sont composées d'un ensemble de processeurs ayant accès à une mémoire commune. La plupart des multiprocesseurs actuellement dans le marché sont des systèmes symétriques (SMP – Symmetric Multi Processing), dans le sens où tous les processeurs ont les mêmes fonctions et sont en compétition pour les ressources système de façon égale.

L'existence d'une mémoire partagée facilite le travail de parallélisation des algorithmes. La mise en œuvre de la coopération entre processeurs est simplifiée car les communications peuvent s'exprimer par des lectures ou écritures de zones de mémoire partagées, se rapprochant de la programmation séquentielle classique. Mais l'accès à la mémoire commune est aussi le point faible de ces architectures. En effet, le couplage fort entre processeur et mémoire fait que l'augmentation du nombre de processeurs augmente aussi la contention d'accès au bus reliant processeurs et mémoires. De ce fait, les architectures à mémoire partagée ne permettent de relier qu'un nombre limité de processeurs, autrement dit elles ne sont pas adaptées pour le parallélisme à grande échelle.

Afin de limiter la charge dans le bus d'interconnexion, les architectures actuelles emploient plusieurs niveaux de mémoires cache, de façon à ce que le temps réel d'accès à une donnée dépende de sa localisation dans la hiérarchie de mémoires. Même avec l'utilisation de ces techniques, les architectures SMP restent aujourd'hui limitées à quelques dizaines de processeurs. Les machines Sun Enterprise10000 sont actuellement les SMP de plus grande taille dans le marché. Malgré cette contrainte d'extensibilité, les architectures SMP se sont beaucoup répandues, en particulier les configurations à deux ou quatre processeurs qui sont aujourd'hui proposées par la plupart des constructeurs de stations de travail.

2.2.Architectures à mémoire distribuée

Les architectures à mémoire distribuée sont constituées d'un ensemble de nœuds, chaque nœud étant composé d'un processeur et d'une mémoire locale. Les nœuds sont connectés entre eux par un réseau de communication, au travers duquel ils peuvent coopérer. Grâce au couplage plus lâche entre processeurs et mémoires, les machines utilisant cette architecture sont plus extensibles à la fois en nombre de processeurs et en capacité mémoire. Cela a permis de construire des machines pouvant employer plus d'une centaine de processeurs. On parle alors d'architectures massivement parallèles (MPP – Massively Parallel Processing).

Les technologies d'interconnexion entre les nœuds sont très variées, tant du point de vue du matériel utilisé que des topologies proposées. Avec l'évolution des technologies standard (FDDI, ATM, etc.), les machines à mémoire distribuée ressemblent de plus en plus à des réseaux d'ordinateurs. Avec un bon rapport prix/performances, les grappes (« clusters » en anglais) construites par l'assemblage de matériel produit en grande série sont devenues très populaires. Si l'absence d'une mémoire commune permet un parallélisme à plus grande échelle, la mise en œuvre d'applications parallèles sur les machines à mémoire distribuée est plus délicate. Aucun support aux accès directs à une mémoire distante n'est a priori fourni par ce type d'architecture, et la coopération entre processeurs est plus coûteuse par rapport aux architectures à mémoire partagée. Ceci est d'autant plus vrai pour les architectures de type grappe. Le programmeur doit par conséquence prendre en compte la localisation des données sur les mémoires locales et s'occuper explicitement de la distribution ou du mouvement de ces données lorsque nécessaire.

2.3.Architectures hybrides

La maturité des multiprocesseurs et l'évolution constante des réseaux d'interconnexion ont mis sur le marché des machines combinant les deux types d'architectures précédentes. Ces machines sont constituées d'un ensemble de nœuds multiprocesseurs reliés entre eux par un réseau d'interconnexion haut débit, l'exemple typique étant l'Origine 2000 fabriquée par SGI. Le modèle de mémoire de ces architectures est mixte : mémoire partagée au sein d'un même nœud, et mémoire distribuée entre les différents nœuds. Certaines machines ont une organisation mémoire où chaque nœud peut accéder à des zones mémoire appartenant à d'autres nœuds, mais la mise en œuvre efficace de ces mémoires virtuellement partagées n'est toujours pas bien maîtrisée. Les architectures hybrides offrent une alternative pour contourner le problème d'extensibilité des machines à mémoire partagée.

Elles ont aussi l'avantage d'une puissance de communication supérieure par rapport aux machines à mémoire distribuée, les premières ayant moins de nœuds que les dernières à une puissance de calcul égale (i.e. un même nombre de processeurs). L'assemblage de multiprocesseurs en réseau s'est montré convenable pour la construction de systèmes de petite à grande taille. Avec l'ample dissémination des machines SMP sur le marché, les grappes et réseaux utilisent des unités à plusieurs processeurs. L'acronyme CluMPs (Clusters of Multi Processors) apparaît pour désigner ces grappes de multiprocesseurs. L'architecture des nouvelles machines massivement parallèles devient aussi très proche d'une grappe de SMP, la différence résidant surtout dans l'utilisation de réseaux propriétaires et dans la préservation de l'image d'un système homogène. Les systèmes de plus grande taille construits sur ce principe accommodent aujourd'hui près de dix mille processeurs, avec des performances dépassant le ter a flops (10^{12}opérations de point flottant par seconde)[17].

Si les architectures hybrides peuvent mener à des performances sans précédent, elles représentent aussi des nouveaux challenges pour le calcul haute-performance. La mise en œuvre d'applications parallèles exploitant efficacement ce type d'architecture reste délicate puisqu'elle doit prendre en compte les différents niveaux de mémoire de la machine. La plupart des modèles de programmation parallèle classiques a été conçue pour les architectures soit à mémoire partagée soit à mémoire distribuée. Ces modèles correspondent mal à une organisation hybride de la mémoire. Le programmeur peut donc être amené à utiliser simultanément deux modèles de programmation différents.

3. Programmation des machines parallèles

L'utilisation des ressources d'une architecture parallèle pour la résolution d'un problème passe par l'extraction du parallélisme d'un algorithme et par la construction d'un programme parallèle exploitant efficacement ce parallélisme. La conception d'un algorithme parallèle est un travail méthodique où il faut prendre en compte successivement un certain nombre de points liés à l'expression du parallélisme.

Nous présentons dans la section 3.1 les principales étapes d'une méthodologie de parallélisation. Les modèles visant la mise en œuvre du parallélisme sont ensuite présentés dans la section 3.2.

3.1.Méthodologie de parallélisation

Une représentation souvent utilisée pour décrire un programme parallèle est celle d'un graphe de tâches. Les nœuds d'un tel graphe sont les tâches d'un programme (des séquences

de calculs) tandis que les arcs représentent les dépendances entre les tâches (il s'agit souvent de communications). Selon cette description d'un programme parallèle, la parallélisation d'un algorithme nécessite le découpage du travail en tâches potentiellement parallèles et la définition des interactions entre elles, suivi d'une distribution de ces tâches sur les processeurs de la machine parallèle.

3.1.1. Choix de la granularité des tâches

Lors du découpage d'un calcul, une question importante concerne la granularité des tâches, c'est à dire la taille de chaque tâche relativement à la taille totale du programme. On entend par "taille d'une tâche" la quantité de calculs associés à cette tâche ne nécessitant pas de communication avec les autres tâches. Dans un découpage à grain fin, le calcul est découpé en un grand nombre de tâches de petite taille. In versement, la décomposition d'un travail en tâches peu nombreuses exigeant beaucoup de calculs caractérise un découpage à gros grain. Une granularité fine offre plus d'opportunités pour l'exploitation du parallélisme et pour l'équilibrage de charge. Mais un grain trop fin entraîne souvent des communications trop fréquentes entre processeurs. En revanche, le découpage à gros grain permet de réduire le temps nécessaire à la gestion du parallélisme, au prix d'une diminution du parallélisme potentiel. Le choix de la granularité doit donc satisfaire un compromis entre le maximum de parallélisme potentiel, un surcoût minimum de gestion du parallélisme, et un bon équilibre de charge entre les processeurs.

En pratique, le choix de la granularité dépend fortement de l'architecture cible et de l'application. Le coût de la communication entre processeurs joue un rôle important dans l'obtention d'un parallélisme efficace : plus ce surcoût est faible dans une architecture, plus elle sera apte à exécuter des tâches de granularité fine. Les architectures à mémoire partagée sont donc idéales pour exploiter le parallélisme à grain fin, tandis que les architectures à mémoire distribuée sont plus adaptées au parallélisme à gros grain. En revanche, les grappes de multiprocesseurs ont l'avantage de permettre l'exploitation simultanée de différents grains de parallélisme.

3.1.2. Expression des interactions entre tâches

La définition des interactions entre les tâches d'un programme, c'est-à-dire l'utilisation par une tâche d'une donnée calculée par autre tâche est aussi importante. Cela établit des contraintes de précédence entre les tâches. Le graphe de tâches agrémenté de cette information est un graphe orienté appelé graphe de précédence. Ce graphe définit un ordre

partiel pour l'exécution des tâches. Les tâches non ordonnées par cet ordre partiel peuvent être exécutées en parallèle.

Une alternative au graphe de précédence consiste à représenter un flot de données sur le graphe. Dans cette représentation, dénommée graphe de flot de données, la précédence entre tâches est induite par les échanges de données. Typiquement, les arcs du graphe correspondent à des opérations de lecture ou d'écriture d'une donnée. Une différence fondamentale entre le graphe de précédence et le graphe de flot de données se trouve au niveau des synchronisations. Dans le cas d'un graphe de précédence, une tâche est prête à exécuter quand toutes les tâches qui la précèdent sont terminées, tandis que dans un graphe de flot de données une tâche est prête si toutes les données auxquelles elle accède en lecture sont disponibles.

La détermination d'un graphe de précédence ou d'un graphe de flot de données dépend de la connaissance du comportement de l'application en terme de phases de calcul et de communication, ainsi que de leurs durées. En fonction de cette connaissance préalable de leur comportement, les applications et les algorithmes parallèles peuvent être classés réguliers ou irréguliers. De manière générale, un algorithme est dit régulier si le graphe de tâches associé est statique, c'est-à-dire, si la durée des tâches de ce graphe est connue avant l'exécution, ou bien peut être prédite à chaque étape d'exécution. À l'in verse, on dit que l'algorithme est irrégulier s'il est impossible de prédire le comportement de l'algorithme avant son exécution, ou si cette prédiction reviendrait à exécuter l'application, ou encore si le comportement de l'algorithme change d'une exécution à l'autre en fonction des données.

3.1.3. Ordonnancement et placement des tâches

Le graphe de tâches constitue un élément central pour l'ordonnancement d'un programme parallèle. On entend par ordonnancement l'attribution d'une date de début d'exécution à chaque tâche composant le graphe. Cette attribution doit être compatible avec le nombre de processeurs disponibles et avec les contraintes de précédence entre les tâches. Cette étape est souvent combinée avec la distribution spatiale ou placement du graphe, c'est-à-dire à l'association d'un site d'exécution(un processeur) pour chacune de ces tâches. L'ordonnancement et le placement d'un graphe de tâches représentant un programme parallèle sont deux aspects fortement liées, tous les deux ayant pour but final l'obtention d'une exécution efficace où les communications entre processeurs sont minimisées et où le taux d'utilisation des processeurs est le plus élevé possible. La distribution des tâches de calcul sur les processeurs doit donc éviter que certains processeurs ne soient sous-utilisés

alors qu'il reste du travail en attente. Cet objectif peut-être vu sous deux optiques légèrement différentes : le partage de charge qui consiste à répartir le travail de façon à ce que tous les processeurs soient actifs à tout instant, et l'équilibrage de charge qui consiste non seulement à garder les processeurs actifs, mais aussi à répartir équitablement le travail entre les processeurs.

L'équilibrage de charge au cours du calcul implique de déplacer des tâches, ce qui suppose des transferts de données entre le processeur surchargé et un processeur moins actif. Cette étape nécessite des communications, dont le coût doit être faible par rapport au gain espéré. En pratique, la décision d'équilibrer la charge est prise en fonction d'un seuil minimal dépendant de la taille des tâches et représentant un compromis acceptable. Ce compromis est généralement plus facile à atteindre lorsque la granularité des tâches est fine et que les données à communiquer sont" peu volumineuses".

Les stratégies d'ordonnancement et de placement sont fortement dépendantes des caractéristiques du graphe de tâches. Dans les cas où le graphe de précédence est complètement déterminé il est possible d'utiliser des heuristiques statiques classiques. Néanmoins, pour un grand nombre d'applications, le comportement du programme dépend des données et le graphe de tâches associé n'est pas prédictible. Ces applications typiquement irrégulières sont caractérisées par la création dynamique d'un nombre important de tâches ayant souvent des grains très différents. La réalisation d'un ordonnancement et d'un placement efficaces pour telles applications n'est donc pas triviale.

3.2.Modèles de programmation parallèle

Un modèle de programmation représente l'abstraction d'une machine dans le but de séparer les aspects concernant le développement de l'application de ceux liés à son exécution effective. Des modèles existent à différents niveaux d'abstraction, du parallélisme totalement explicite (où son expression est complètement à la charge du programmeur) au parallélisme implicite (parallélisation automatique).

Les modèles explicites proches des architectures sont assez répandus pour les applications visant la haute-performance, mais la programmation basée sur ces modèles est plus difficile car tous les détails de la parallélisation sont apparents. Dans la suite nous faisons un tour d'horizon sur les modèles qui s'adaptent le mieux à chaque type d'architecture, puis nous introduisons quelques modèles à haut niveau d'abstraction. Cette présentation est loin d'être exhaustive.

3.2.1. Programmation des architectures à mémoire partagée

Sur architecture à mémoire partagée, la programmation parallèle est souvent fondée sur des techniques de multiprogrammation légère (« multithreading »)[14]. Selon ce modèle, le programme parallèle est composé de multiples flots d'exécution (fils d'exécution) concurrents manipulant des données placées dans une mémoire commune. La notion de fil d'exécution ou processus léger est employée depuis longtemps au sein des systèmes d'exploitation afin d'exprimer des tâches concurrentes au sein d'un même programme. Si dans une architecture conventionnelle les processus légers sont en compétition pour l'utilisation du processeur, dans une machine SMP ils peuvent réellement s'exécuter en parallèle. Par définition, les processus légers appartenant à un même processus système partagent des zones mémoire allouées à ce processus, d'où l'emploi naturel de la multiprogrammation légère sur des machines SMP.

Étant donné que les lectures et écritures dans la mémoire commune se font en parallèle, la difficulté de ce modèle de programmation est de garantir la cohérence d'accès aux zones de mémoire partagée afin de préserver la sémantique correcte du programme. Pour cela le programmeur doit recourir à des primitives de synchronisation comme les verrous ou les variables de condition. La programmation se fait par l'intermédiaire de bibliothèques de processus légers qui sont disponibles dans la plupart des systèmes. Un jeu de primitives standard pour des bibliothèques offrant la gestion et la synchronisation de processus légers est définit par la norme POSIX[18]. La multiprogrammation légère peut aussi être offerte au niveau du langage de programmation, comme dans Java[19].

Une approche plus récente pour la programmation des SMP est le standard OpenMP[20]. Il consiste essentiellement en un ensemble de procédures et directives que le programmeur ajoute à un programme écrit en Fortran ou C/C++ a fin d'aider le compilateur à prendre des décisions de parallélisation. La multiprogrammation légère est une notion intrinsèque dans ce standard, mais les processus légers sont confinés au support exécutif de OpenMP, le programmeur n'ayant pas à les gérer lui-même.

3.2.2. Programmation des architectures à mémoire distribuée

Dans une architecture à mémoire distribuée, la coopération entre processeurs se fait classiquement par échange de messages. Le programme parallèle est organisé comme une collection de processus s'exécutant sur un ensemble de processeurs, chaque processus possédant une zone de mémoire privée. L'échange de données entre processus n'est possible que par des opérations d'envoi et réception explicites. L'approche de programmation la plus

répandue est d'utiliser une bibliothèque d'échange de messages telle que PVM[21] ou MPI[22]. Cette dernière est en fait un standard pour la programmation par échange de messages, visant à la fois la haute performance des communications et la portabilité des programmes parallèle sre posant sur ce paradigme. Des bibliothèques suivant le standard MPI sont disponibles sur un grand nombre de plates-formes actuellement, et certains constructeurs en proposent des versions hautement optimisées pour une architecture en particulier.

La possibilité de mettre au point des applications efficaces sur des architectures extensibles est un point fort du paradigme d'échange de messages, mais cela se fait au prix d'une plus grande complexité de programmation parallèle. Comme le temps de transmission des données est grand par rapport à la puissance de calcul des processeurs, il faut chercher à réduire le plus possible les communications par un choix minutieux de la granularité des tâches et une distribution intelligente des données sur les mémoires locales.

3.2.3. Programmation des grappes de multiprocesseurs

En ce qui concerne les architectures parallèles ayant un modèle de mémoire hybride, le paysage des modèles de programmation est moins bien défini que celui des architectures purement à mémoire partagée ou à mémoire distribuée. Selon l'abstraction de l'organisation mémoire offerte au programmeur, deux approches principales sont envisageables pour la programmation de CluMPs : un modèle de programmation uniforme, où le programmeur ne voit qu'un seul modèle mémoire (partagée ou distribuée), ou un modèle de programmation hybride, qui combine la multiprogrammation légère et l'échange de messages. Pour l'unification du modèle de mémoire, une alternative consiste à utiliser des environnements de programmation fournissant la fonctionnalité d'une mémoire virtuellement partagée. Les performances d'un tel modèle sont néanmoins très dépendantes du protocole de gestion de la cohérence de la mémoire. L'autre alternative est de programmer les CluMPs comme des machines à mémoire distribuée.

On trouve ainsi des bibliothèques d'échange de messages mises au point pour une communication efficace à l'intérieur des nœuds. Le modèle de programmation hybride, à son tour, constitue une des premières approches pour la programmation des CluMPs. En effet, l'association de la multiprogrammation légère à la communication par échange de messages suscitait déjà beaucoup d'intérêt avant que les CluMPs deviennent populaires, par exemple par les possibilités de masquer les temps de communication par des calculs utiles et de gérer simplement et élégamment l'indéterminisme des communications. Parmi les bibliothèques construites surce principe on peut citer ATHAPASCAN- 0 , Pmet Nexus.

3.2.4. Programmation indépendante d'architecture

Afin de faciliter la compréhension et le développement d'applications parallèles, mais aussi d'augmenter la portabilité de ces applications, certains modèles de programmation travaillent avec une plus grande abstraction de l'architecture parallèle. Parmi les approches de plus haut niveau on trouve la parallélisation automatique ou semi-automatique, qui vise la génération d'un code parallèle à partir d'un programme séquentiel, par l'intermédiaire d'un compilateur capable d'extraire le parallélisme présent dans un tel programme. La génération du code parallèle est souvent guidée par des directives de compilation rajoutées dans le programme source afin de définir, par exemple, un partitionnement des données du programme. Une telle technique est employée dans le langage HPF (High Performance Fortran), entre autres. Cette approche est pratique pour le programmeur, mais les difficultés liées à la détection automatique du parallélisme et à la génération de codes efficaces sur les différentes architectures parallèles font que les outils de parallélisation automatique sont moins généraux et par conséquence moins répandus que les outils de programmation parallèle explicite. En effet, la parallélisation automatique est actuellement plus efficace sur des architectures à mémoire partagée que sur mémoire distribuée, et les bons résultats sont surtout obtenus pour des applications constituées de nids de boucles et pour lesquelles le parallélisme ne dépend pas des données d'entrée.

Dans un niveau intermédiaire entre la parallélisation explicite et la parallélisation automatique, on trouve des paradigmes permettant d'abstraire certains aspects de la machine parallèle. Les environnements offrant la fonctionnalité d'une mémoire virtuellement partagée en sont un exemple : ils permettent d'abstraire le réseau de communication en fournissant des mécanismes de partage de données entre processeurs, occultant ainsi les communications inhérentes à une exécution sur architecture distribuée. La granularité des données partagées est variable : certains systèmes comme Tread Marks et SAM offrent un espace d'adressage unique de la mémoire, l'unité de partage étant le mot mémoire élémentaire. L'aspect critique de cette approche pour la programmation parallèle réside dans sa mise en œuvre efficace sur une architecture distribuée, étant donnée que la préservation de la cohérence de la mémoire virtuellement partagée génère du trafic sur le réseau. Les systèmes offrant le partage au niveau de l'objet ont tendance à être plus efficaces sur architecture distribuée, puisque la granularité des données partagées est plus grosse et mieux adaptée à l'application [23].

Finalement, on trouve des systèmes permettant de séparer l'expression du parallélisme du placement réel des données et des calculs sur la machine. Il s'agit ici d'une abstraction des processeurs composant l'architecture parallèle : le programmeur définit explicitement les

tâches du programme parallèle et les dépendances entre elles, sans pour autant prendre en compte l'architecture sur laquelle aura lieu l'exécution. Le système exécutif assure l'ordonnancement automatique des tâches et une exécution correcte de l'application basée sur une analyse du graphe de tâches ou du graphe de flot de données. Le programmeur ne gère pas explicitement les communications entre les tâches, ce qui facilite grandement la mise en œuvre parallèle.

4. Multiprogrammation légère pour le calcul parallèle

Quand l'algorithmique du problème à paralléliser peut être conçue de manière à isoler un grand nombre de tâches, il est naturel d'envisager une exécution de celle-ci en utilisant des processus différents. Une technique adaptée à la gestion de ces processus, tant pour les architectures à mémoire partagée que pour les architectures à mémoire distribuée, est la multiprogrammation légère. Pour les secondes, ces processus sont alors utilisés conjointement avec une bibliothèque de communication.

4.1.Les processus légers en bref

Un processus léger (ou PL) est une abstraction permettant d'exprimer des multiples flots d'exécution indépendants au sein d'un processus système (e.g. UNIX, appelé processus lourd). Les différents flots d'exécution partagent le contexte de mémoire du processus lourd, leur gestion est donc moins coûteuse que celle des processus système. En effet, la gestion des processus légers nécessite seulement la manipulation de leur contexte d'exécution, tandis que les opérations sur les processus lourds demandent aussi la gestion de zones mémoire et de ressources (fichiers ouverts, verrous, etc.) allouées par le système. Par exemple, un changement de contexte pour un processus léger est de 10 à 1000 fois plus rapide que pour un processus lourd.

Les opérations usuellement disponibles pour la programmation avec des processus légers sont la création (sur le même processeur ou sur un autre processeur), la destruction (souvent l'auto-destruction), la communication entre PL (par messages, appels de procédure à distance, etc.) et la synchronisation (par sémaphores, verrous d'exclusion mutuelle, barrière de synchronisation, partage de données, etc.). La norme POSIX définit un standard pour les opérations manipulant des processus légers et des implantations sont proposées par la plupart des grands constructeurs d'ordinateurs.

Bien qu'au niveau de l'interface de programmation les bibliothèques de Pl offrent souvent des fonctionnalités similaires, leur mise en œuvre diffère sensiblement. Les différents

choix d'implantation de ces bibliothèques et de leur intégration avec le système d'exploitation sont étudiés en détail dans [24] et [25], entre autre. Nous ne donnons ici qu'un aperçu des différentes techniques proposées au fil du temps. Ainsi, par exemple :

- certaines bibliothèques s'appuient sur des "PL système", qui sont connus du système d'exploitation sous-jacent. D'autres reposent sur des "PL utilisateurs", qui sont plus "légers" que les précédents. Ils ne sont pas connus du système d'exploitation mais seulement de l'environnement de programmation parallèle qui offre ces PL utilisateurs
- la multiprogrammation des PL peut être implantée avec différentes formes de préemption : préemption uniquement quand le PL est bloqué (e.g. en attente d'un message ou d'un sémaphore), préemption à la fin d'un quantum, etc.
- la politique d'ordonnancement des PL sur chaque processeur peut varier : premier arrivé-premier servi, utilisation de priorités, etc.

Lorsqu'un environnement de programmation parallèle utilise conjointement une bibliothèque de PL et une bibliothèque de communication (comme PVM ouMPI), il doit y avoir compatibilité (plus ou moins forte) entre les deux bibliothèques :

- si la bibliothèque de communication est « thread-safe », elle peut être utilisée de façon concurrente par plusieurs PL ;
- si une opération de communication est « thread-synchronous », elle peut bloquer le PL qui l'exécute sans bloquer les autres PL dans le même contexte ;
- si la bibliothèque de communication est « thread-aware », elle connaît la multiprogrammation des PL.

4.2.Le bénéfice à tirer des processus légers

Prenons l'exemple général d'un programme composé de tâches indépendantes(voir Figure 24). L'utilisation de processus légers simplifie la programmation, car la coordination entre tâches est réduite. Les tâches sont exécutées de façon concurrente ou en parallèle et, si un processus léger est bloqué (par exemple, en attendant une lecture d'un fichier), les autres processus légers peuvent continuer. Le processeur est ainsi mieux utilisé.

Les PL constituent une technique naturelle pour l'implémentation des tâches d'une application parallèle. Cette vision d'un programme parallèle comme un ensemble de tâches/PL interagissant a plusieurs intérêts :

- elle permet de décrire l'application indépendamment de la machine cible, comme un ensemble de tâches multiprogrammées s'exécutant sur un nombre arbitraire de

processeurs (mais en pratique inférieur au nombre de tâches potentiellement parallèles) ;

- elle simplifie la programmation des applications. Le programmeur n'a pas à gérer lui-même le masquage des attentes et des communications par du calcul, qui est pris en charge par la multiprogrammation de chaque processeur.

Si l'environnement dispose d'un ordonnanceur global, celui-ci prend aussi en charge l'allocation des tâches/PL aux processeurs (statiquement ou dynamiquement) afin que tous les processeurs aient des tâches actives. Il faut noter que la structuration d'un programme sous forme de graphe de tâches est la bonne structure pour les algorithmes d'ordonnancement.

Les PL communicants sont donc le mécanisme d'abstraction d'une machine parallèle (à mémoire commune ou distribuée) permettant une programmation flexible et réactive d'un programme parallèle composé de tâches.

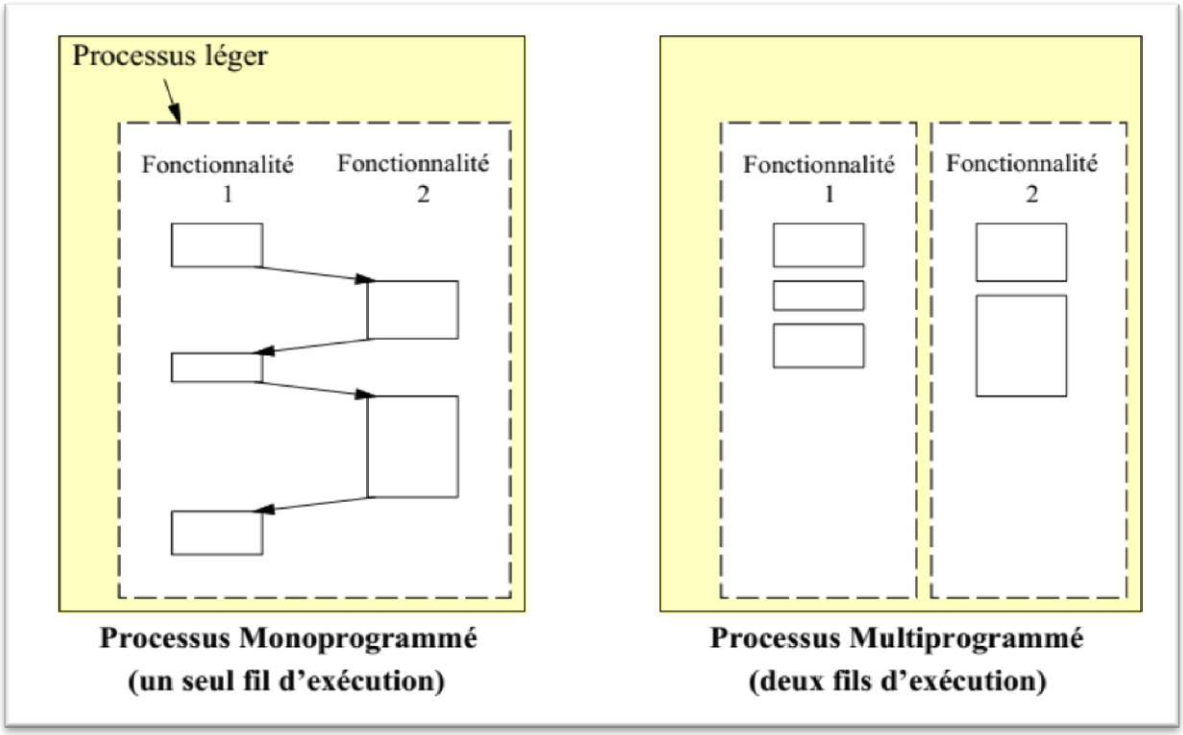


Figure 24. *Programmation classique vs. Multiprogrammation d'un programme composé de tâches indépendantes*

5. OpenMP (Open Multi-Processing)

OpenMP (Open Multi-Processing) est un outil de programmation parallèle en langage C/C++ ou Fortran. Il est basé sur un modèle multithreading pour des architectures à mémoire partagée. Il se présente sous la forme de directives de compilation insérées dans le code source (des pragmas), des variables d'environnement et d'une bibliothèque logicielle permettant de développer rapidement des applications parallèles à petite granularité en restant proche du code séquentiel.

Un programme OpenMP est fait de régions séquentielles ainsi que de régions parallèles qui peuvent être exécutées par plusieurs threads en parallèle sur différents cœurs. Le compilateur se charge de la création des threads, leurs synchronisations et les communications entre eux, ce qui facilite fortement le travail des programmeurs. L'injection du parallélisme avec OpenMP peut se faire sur trois niveaux :

a- Parallélisations d'une région parallèle du code sur différents threads comme indiqué par la Figure 25.

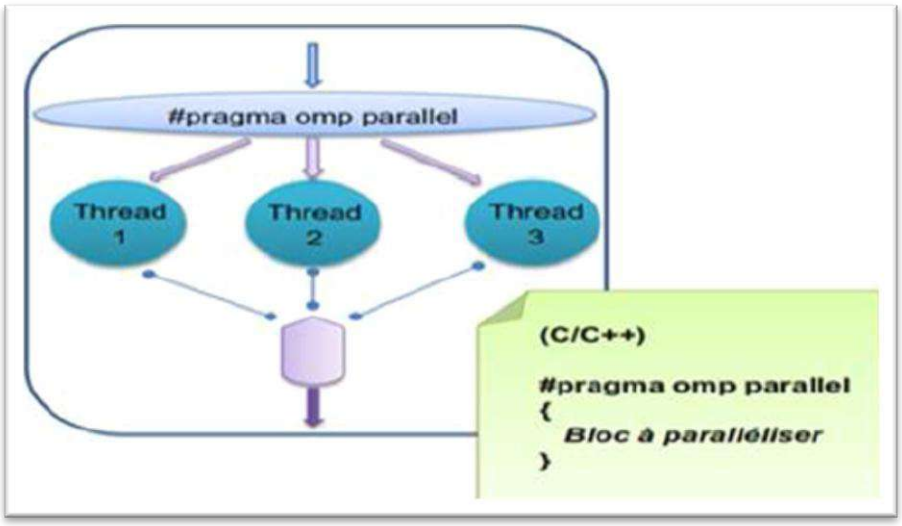


Figure 25.*Parallélisation d'une région parallèle avec OpenMP*

Le nombre de threads est définit par la variable d'environnement :Set OMP_NUM_THREADS = x.

En général, le nombre de threads est égal au nombre de processeurs (ou cœurs).

b- Parallélisation des boucles en répartissant les itérations entre les threads comme l'indique la Figure26.

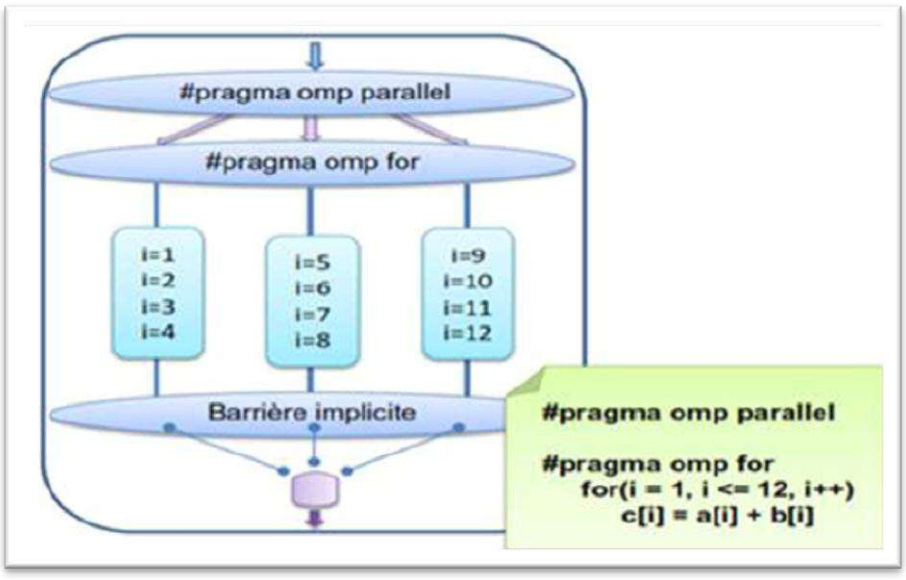


Figure 26.*Parallélisation des boucles avec OpenMP*

c- Parallélisation des sections indépendantes sur différents threads comme illustré par la Figure 27.

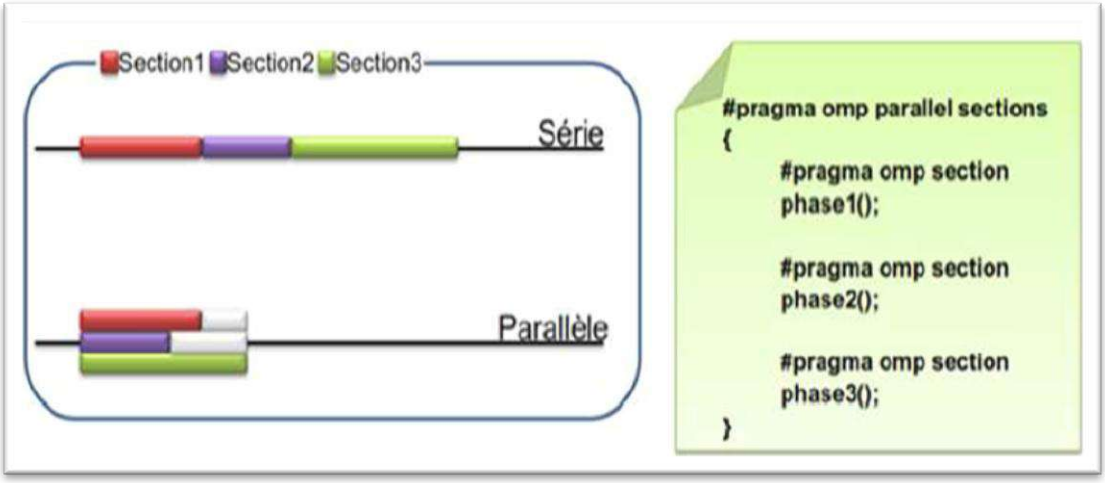


Figure 27.*Parallélisation des sections avec OpenMP*

Il faut cependant être très prudent avec OpenMP. En effet, il faut veiller à ne pas créer de « Data race », c.à.d. des accès simultanés à des variables partagées par différents threads.

6. Conclusion

Dans ce chapitre nous avons exploré les différents outils de programmation parallèle. Dans un système à plusieurs processeurs, les données doivent être accessibles par tous les processeurs, par conséquent différentes configurations de mémoire sont possibles pour placer les données vis-à-vis les processeurs, on parle principalement de systèmes à mémoire partagée, à mémoire distribuée et systèmes hybrides. Les architectures à mémoire partagée offre une meilleure communication inter-processeurs, tandis que les architectures à mémoire distribuée permet l'interconnexion d'un nombre élevé de processeurs. Une architecture hybride permet de réunir les avantages des architectures à mémoire partagée et à mémoire distribuée en une même architecture. Pour des raisons de simplicité et de flexibilité, nous avons retenu le choix des architectures à mémoire partagée pour notre application, d'ailleurs c'est l'architecture opté pour la pluparts des ordinateurs personnels. Une description détaillée sur la programmation par OpenMP est donnée à la fin de ce chapitre.

Chapitre III : Détecteurs de contour

1. Introduction

Nous avons vu dans le premier chapitre que le partitionnement d'un bloc se fait en quad-tree, et qu'un test sur toutes les formes possibles doit être fait afin d'optimiser la fonction de distorsion. Comme contribution à l'optimisation du temps d'encodage, nous proposons de ne pas partitionner les blocs à caractère homogène (pas trop de texture). Pour avoir l'information sur l'homogénéité d'un bloc nous avons besoin d'un détecteur de contour qui est l'objet de ce chapitre.

La détection de contour et la segmentation des images sont probablement les domaines qui ont reçu la plus grande attention de la part de la communauté de traitement des images. On a pu recenser ainsi plus de 4000 références sur ce sujet depuis 1970. On ne sera donc pas surpris de trouver ici des méthodes extrêmement variées, s'appliquant parfois à des domaines bien précis. Ceci est en grande partie dû à la nature très intuitive du contour qui apparaît très naturellement comme l'indice visuel idéal dans la plus grande partie des situations.

Très schématiquement, les contours sont les lieux de variations significatives de l'information niveaux de gris (il y a très peu de travaux sur les contours dans les images couleurs ou multi-spectrales). Dans cette approche, on suppose que l'image est une mosaïque de régions parfaitement homogènes. C'est à dire que les contours recherchés sont de type créneaux. De plus, la transition étant stricte, un contour doit être une chaîne de pixels d'épaisseur 1. Cette restriction sur la nature du contour a été imposée dans un premier temps pour des raisons de formalisation mathématique. Il est possible de construire des processus capables d'extraire d'autres types de contours comme par exemple des vallées ou des toits. Cependant, il n'existe pas à l'heure actuelle de processus complet et général qui pourrait extraire tous les types de contour.

2. La détection de contour par les méthodes dérivatives

La notion de contour étant reliée à celle de variation, il est évident qu'une telle définition nous amène tout naturellement vers une évaluation de la variation en chaque pixel. Une variation existera si le gradient est localement maximum ou si la dérivée seconde (à définir dans un espace bi-dimensionnel) présente un passage par zéro. Les principaux algorithmes connus (Sobel, Prewitt, Kirsh, Canny, Dériche, …) se focalisent sur ce premier aspect du contour. Il existe moins de travaux sur la formalisation de la deuxième partie consistant à passer d'une mesure locale de variations à des chaînes de points d'épaisseur 1.

C'est pourtant cette deuxième partie qui fait souvent la différence et la qualité visuelle d'un résultat.

Les méthodes dérivatives s'appuient sur la constatation que les contours d'image sont traduits généralement par les transitions rapides de l'image, et que les variations lentes seront éliminées par dérivation. Les contours des objets dans des images de dimension quelconques (images naturelles 2D, images médicales 2D et 3D) correspondent le plus souvent aux extréma locaux du gradient ou aux zéro du laplacien de la fonction de niveaux de gris.

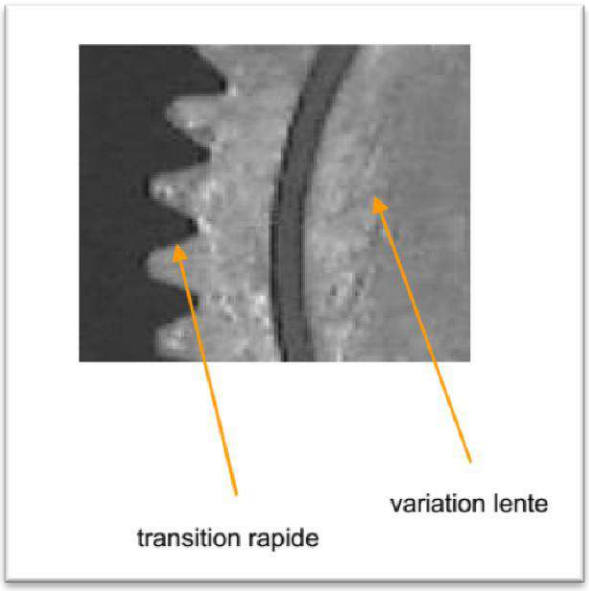


Figure 28 *Illustration de la notion de contour*

2.1.Rappels sur le filtrage linéaire

Filtrer signifie convoluer une image I(x,y) avec une fonction f(x,y) appelée réponse impulsionnelle (ou opérateur de convolution) du filtre. Dans le cas continu, l'image filtrée est donnée par **:**

$$\boldsymbol{I_f(x,y) = (f * I)(x,y) = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} f(x',y')I(x-x',y-y')dx'dy'} \qquad \textbf{(3.1)}$$

Dans le cas discret et pour prendre l'exemple simplifié d'une image carrée, les domaines de $\boldsymbol{I}$ et de $\boldsymbol{f}$ sont bornés. Le domaine de $\boldsymbol{I}$ est **[-N/2, +N/2]²** si **N** désigne la taille de l'image et le domaine de $\boldsymbol{f}$ est **[-K/2,+K/2]²** avec nécessairement **K≤N**. La convolution s'écrit alors :

$$\begin{aligned} I_f(x,y) &= (f * I)(x,y) \\ &= \sum_{i'=-K/2}^{K/2}\sum_{j'=-K/2}^{K/2} f(i-i';j-j')I(i',j') \end{aligned} \qquad \textbf{(3.2)}$$

On notera que le filtrage linéaire consiste simplement à remplacer chaque niveau de gris par une combinaison linéaire des niveaux de gris des points voisins.

Le rehaussement

Un opérateur de rehaussement vise à remplacer le pixel central par la somme des différences avec ses voisins. Un exemple de filtre de rehaussement ci-dessous :

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Comme pour le filtre de moyenne, il est nécessaire de normaliser le résultat. Cependant, le résultat peut être négatif, il convient alors d'y ajouter une constante (128 dans le cas présent).

2.1.1. Les filtres étroits

Ce sont des filtres de dérivée première, et l'on cherche alors le maximum de leur réponse. Leur prototype est le filtre de gradient qui se calcule généralement sur 3 points.

Le filtre de gradient

Le gradient, en un pixel d'une image numérique, est un vecteur caractérisé par son amplitude et sa direction. L'amplitude est directement liée à la quantité de variation locale des niveaux de gris. La direction du gradient est orthogonale à la frontière qui passe au point considéré.

La méthode la plus simple pour estimer un gradient est donc de faire un calcul de variation mono dimensionnelle, i.e. en ayant choisi une direction donnée. On a alors le schéma suivant :$G_d(x,y) = (I * W_d)(x,y)$.

Où $\boldsymbol{W_d}$ désigne l'opérateur de dérivation dans la direction ***d*** et ***** le produit de convolution.

$$G_d(x,y) = \sum_{i=-m}^{m} \sum_{j=-n}^{n} I(x+i, y+i) W_d(i,j) \qquad \textbf{(3.3)}$$

Dans cette version discrète, la taille de cet opérateur est donnée par le couple (***m,n***). Sauf cas très particulier, on utilise toujours **m= n**.

Il existe de très nombreux opérateurs différents (Roberts, Sobel, Prewitt, Kirsch, …) qui ont globalement les mêmes propriétés. Nous allons les détailler dans les pages suivantes

Le gradient étant un vecteur, l'approche la plus classique pour estimer le gradient consiste à choisir deux directions privilégiées (naturellement celles associées au maillage, i.e. ligne et colonne) orthogonales, sur lesquelles on projette le gradient. A partir de deux calculs identiques à celui présenté ci-dessus, on peut donc obtenir une connaissance parfaite du gradient. Le gradient d'une image se calcule comme suit :

$$I_x(x,y) = \frac{\partial I_f(x,y)}{\partial x} et I_y(x,y) = \frac{\partial I_f(x,y)}{\partial y} \qquad \textbf{(3.4)}$$

En chaque point **(x, y)** de l'image, on peut donc calculer le vecteur gradient. Sa direction maximise la dérivée directionnelle et sa norme est la valeur de cette dérivée. On obtient la dérivée de ***I*** dans une direction quelconque $\vec{d}$ à partir des deux dérivées directionnelles définissant le gradient I_x et I_y de la manière suivante : $L_{\vec{d}}=\nabla I \cdot \vec{d}$

En développant la formule de Lagrange au premier ordre, la dérivée en un point ***x*** d'une fonction ***f*** s'obtient par l'approximation suivante : $f'(x) \approx \frac{f(x+1)-f(x-1)}{2}$ **(3.5)**

Appliquée à une image, considérée comme une fonction de deux variables, on peut définir deux dérivées partielles, suivant ***x***(colonnes) et suivant ***y***(lignes). Les masques correspondants figurent sur la Figure 6.

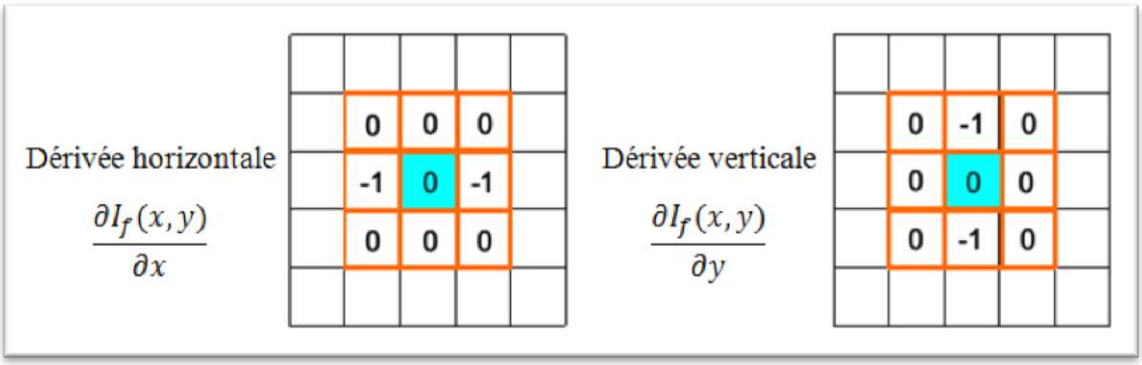


Figure 29 *Le filtrage gradient*

Pour la dérivation accentuant le bruit (pixels parasites de répartition aléatoire), des filtres dérivés, plus robustes, ont été proposés :

- Filtre de Roberts :

Calculé sur 4 points, ce filtre consiste en l'application du double masque suivant :

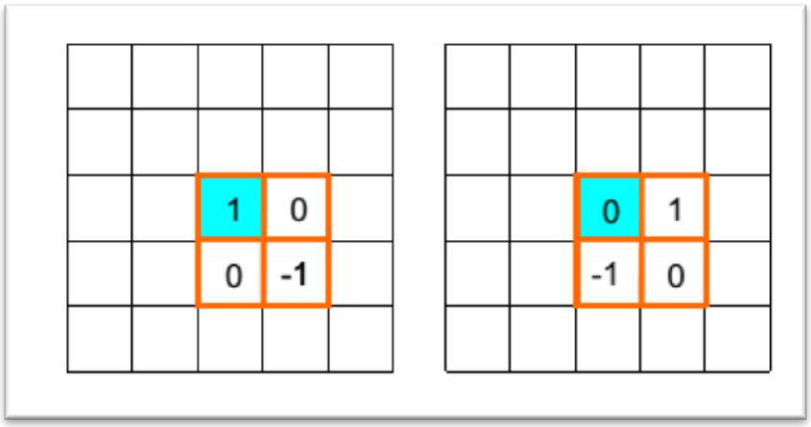


Figure 30. *Masques de Roberts*

En pratique, le filtre de Roberts se calcule de la façon suivante :

Roberts[x][y] = abs(I[x][y]-I[x+1][y+1]) + abs (I[x+1][y]-I[x][y+1]) **(3.6)**

- Le filtre de Prewitt

Calculé sur 9 points, ce filtre effectue une moyenne locale sur 3 points en même temps que la dérivation. Il est défini par le double masque suivant (normalisé par un facteur de 1/3) :

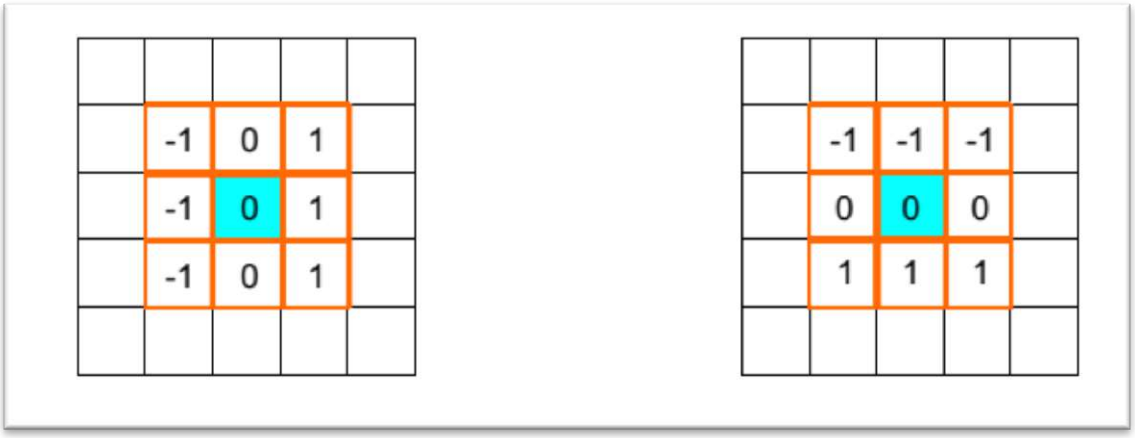


Figure 31. *Masques de Prewitt horizontal et vertical*

- Le filtre de Sobel

Egalement calculé sur 9 points, ce filtre est une variante du filtre de Prewitt mais permet de privilégier le calcul suivant certaines directions (horizontale, verticale, obliques). Les filtres sont donc les suivants (Figure 32)

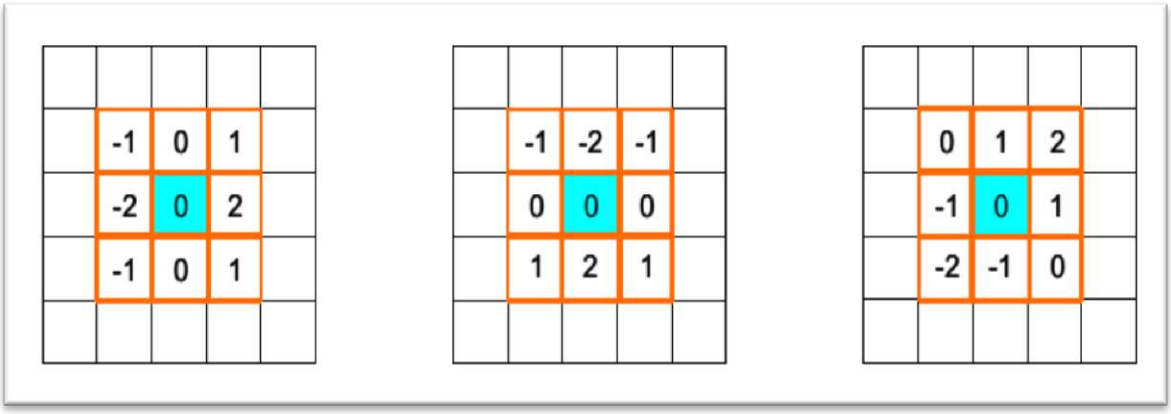


Figure 32. *Masques de Sobel horizontal, vertical et oblique*

Il est important de noter que l'utilisation d'un facteur de normalisation (1/4 en ce qui concerne ces filtres) doit être introduit si l'on veut rester rigoureux.

- Le filtre de Kirsch

Il s'agit de la dernière variante « classique du filtrage gradient. Le facteur de normalisation à utiliser est ici 1/15.

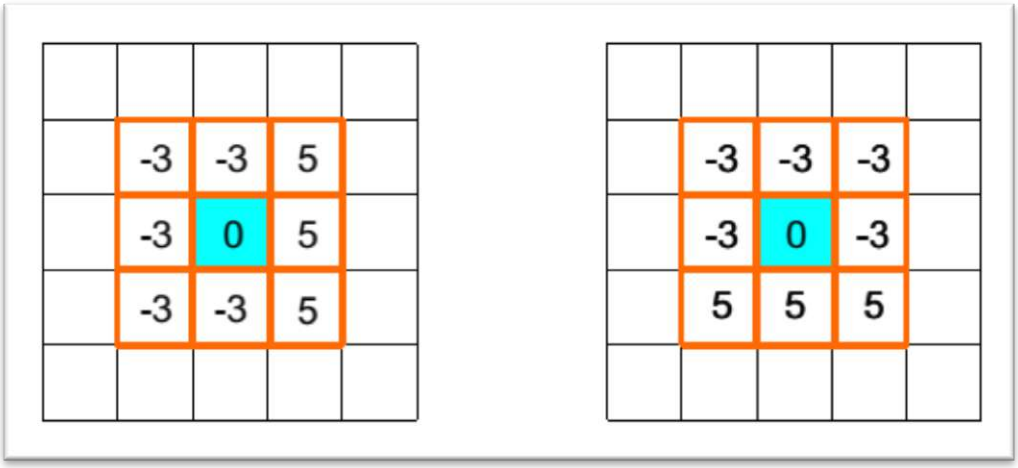


Figure 33. *Masques de Kirsch*

Ces filtres sont certes moins précis que le filtre gradient « pur » mais sont plus fiables de parleur robustesse. Il reste néanmoins utile de les faire précéder de filtres locaux débruiteurs, tels le filtre médian ou le filtre de Nagao.

Les contours obtenus à l'aide des filtres de gradient sont toutefois d'une qualité médiocre et ne peuvent en règle générale être utilisés tels quel car ils sont :

- Bruités
- Epais
- Interrompus
- non fermés

Ces défauts peuvent être compensés par des filtrages ultérieurs relativement simples(seuillages, filtrages morphologiques et logiques) mais dont l'enchaînement est souvent délicat.

Ces méthodes se caractérisent par ailleurs par une grande rapidité et un faible coût, du fait de l'aspect local de ces filtres.

2.1.2. Les filtres larges

Une frontière est un lieu de variation. On peut donc la localiser par la recherche du maximum de la dérivée première (approche basée sur le calcul du gradient). On peut aussi rechercher le passage par zéro de la dérivée seconde (également appelé ***zéro-crossing***).

Dans le cas d'une image, il n'existe pas une dérivée seconde unique mais 4 dérivées partielles(selon $\mathbf{x^2}$, $\mathbf{y^2}$, **xy** et **yx**). En pratique, on lève cette ambiguïté en ayant recours à l'opérateur **Laplacien** qui fait la somme des deux dérivées partielles principales.

$$\nabla^2 I = \frac{\partial^2 I}{\partial x^2} + \frac{\nabla^2 I}{\partial y^2} \qquad \textbf{(3.7)}$$

Les filtres larges recherchent donc les passages par zéro de la dérivée seconde du niveau de gris. Ils sont précédés par des filtres passe-bas à très large support (10x10 ou 30x30 pixels)car ils sont très sensibles au bruit.

La dérivée seconde peut être obtenue par un **Laplacien** ou approximée par des différences de filtres de dimensions différentes.

Avec la même approximation que pour les filtres étroits, on peut définir les dérivées partielles du second ordre grâce aux filtres suivants :

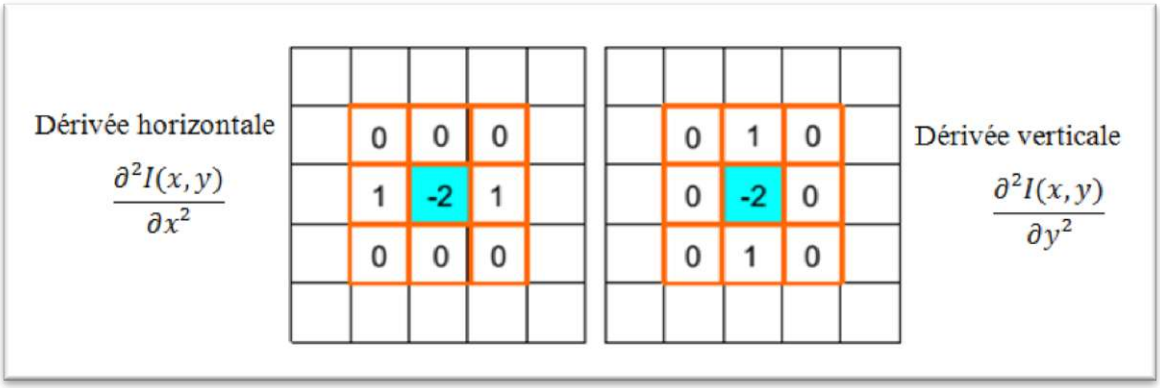


Figure 34. *Le filtrage par dérivée seconde*

L'opérateur **Laplacien** donne une approximation directe de la somme des dérivées secondes, ce qui peut être obtenu avec une matrice qui est la somme des deux précédentes (Figure 35a). Des variantes de ce filtre existent, basées sur d'autres approximations (Figures 35b et 35c).

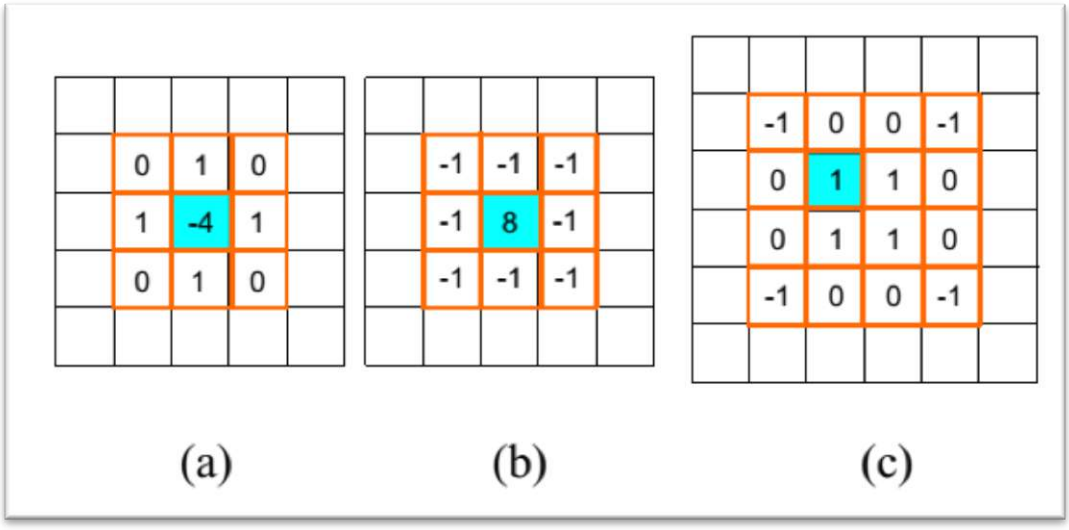


Figure 35. *Masques d'approximation du Laplacien.*

Bien que très sensible au bruit le **Laplacien** est également utilisé pour détecter des contours, caractérisés par le passage à zéro de la dérivée seconde. La Figure 36 illustre la détection de contours selon les méthodes de Sobel et du Laplacien :



Figure 36. *Image originale, filtrage de Sobel et filtrage Laplacien.*

Comme nous l'avons vu précédemment, les filtrages gradient et surtout le Laplacien sont très sensibles au bruit, ce qui justifie l'emploi de filtres dé-bruiteurs. A titre d'exemple, la Figure 37illustre l'emploi de tels filtres en prétraitement.

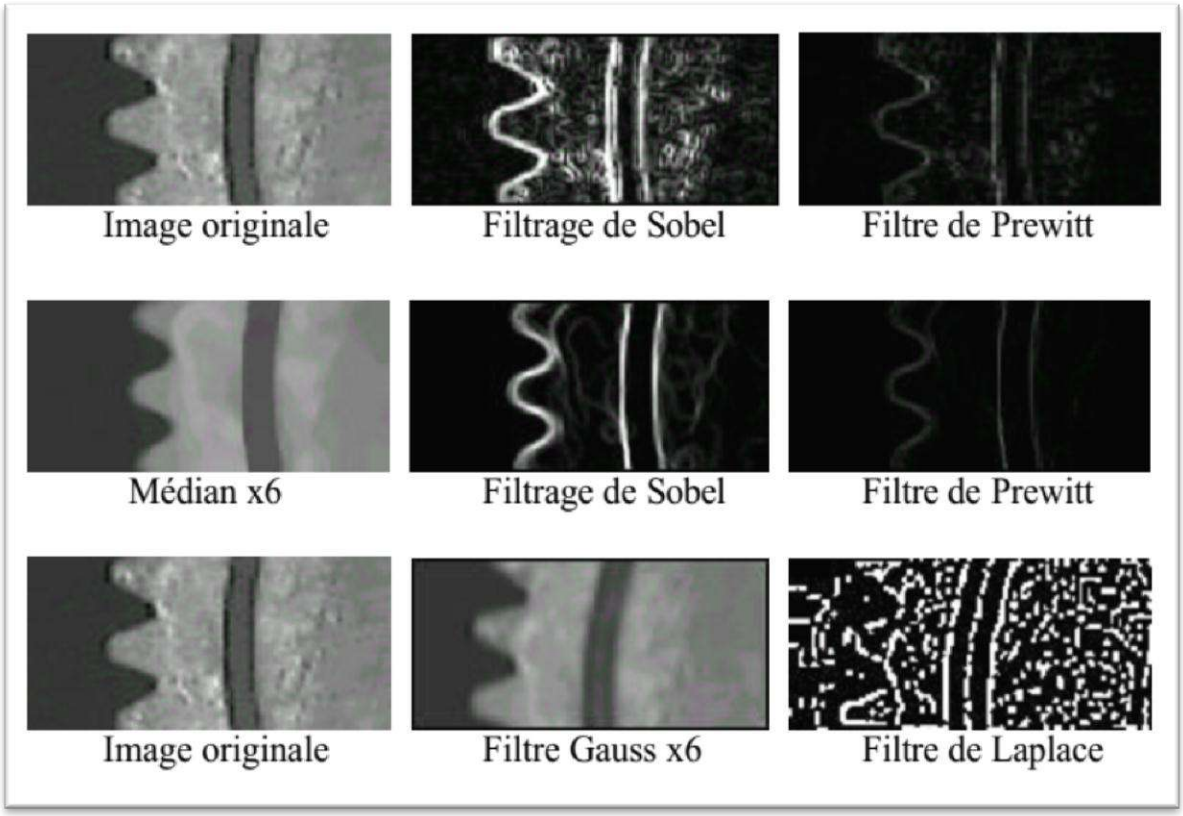


Figure 37. *Dé-bruitage et détection de contours*

Afin de limiter les réponses dues au bruit de l'image ***I***, elle est préalablement filtrée par un filtre ***G***. On obtient alors, grâce aux propriétés de l'opérateur produit de convolution :

$$\nabla^2(I * G) = (\nabla^2 I) * G = I * (\nabla^2 G) \qquad \textbf{(3.8)}$$

Les opérateurs de filtrage et de dérivation se font donc en une seule étape de calcul. Le plus souvent, on fait appel à un filtrage Gaussien :

$$G(r) = -\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{r^2}{2\sigma^2}\right), \quad r = \sqrt{x^2 + y^2} \qquad \textbf{(3.9)}$$

Dont le Laplacien est :

$$\nabla^2 G(r) = -\frac{1}{\pi\sigma^4}\left(1-\frac{r^2}{2\sigma^2}\right)exp\left(-\frac{r^2}{2\sigma^2}\right) \quad \textbf{(3.10)}$$

plus connu sous le nom de « chapeau mexicain » illustré sur la Figure38.

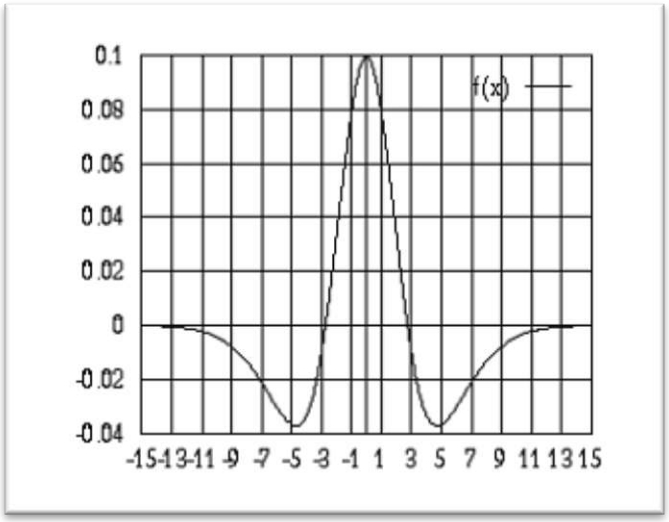


Figure 38. *Laplacien de la fonction de Gauss*

Le paramètre σ sert à régler la résolution à laquelle les contours sont détectés. Son choix est très difficile a priori car il est peu probable qu'une valeur unique permette de représenter efficacement toutes les résolutions présentes dans une image (surtout pour les images naturelles).

Dans le cas discret, la plus simple approximation du Laplacien d'une Gaussienne est le filtre suivant connu sous le nom de masque Laplacien :

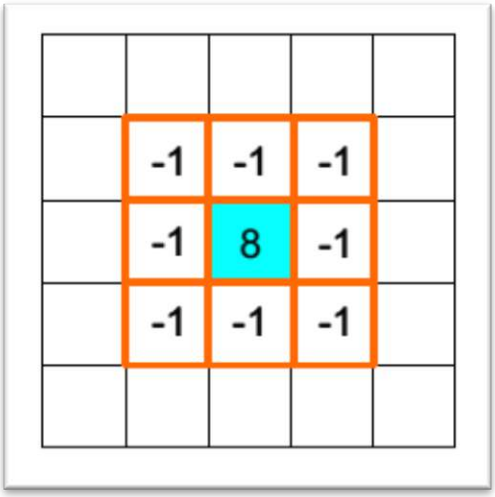


Figure 39. *Masque Laplacien*

3. L'algorithme de l'homogénéité

Les méthodes présentées dans la section 2 reposent sur l'utilisation d'un filtre (convolution) pour détecter les contours, ce qui nécessite de faire des opérations de multiplication couteuses en termes de temps de calcul. Pour des contraintes d'exécution en temps réel nous avons opté pour le choix d'un algorithme simple pour la description d'un bloc (homogène ou texturé). Cet algorithme utilise uniquement la soustraction qui est moins pénalisante par rapport à la multiplication.

Pour un bloc d'une image, on lui fait correspondre une matrice des différences. Dans une première étape, la différence entre un pixel et son voisinage (on a 8 pixels de voisinage) est évaluée, ensuite seule la différence maximale sur les huit différences est considérée et affectée à la matrice finale aux mêmes indices que le pixel considéré. Un seuillage facultatif peut être appliqué aux coefficients de la matrice résultante. Noter bien que les lignes et les colonnes extrêmes () du bloc de l'image ne sont pas considérées puisqu'elles ne possèdent pas de voisinage la première et la dernière ligne, la première et la dernière colonne, et seront remplacer par des zéros dans la matrice finale. Un exemple de calcul pour un bloc de 4x4 est illustré dans la Figure 40.

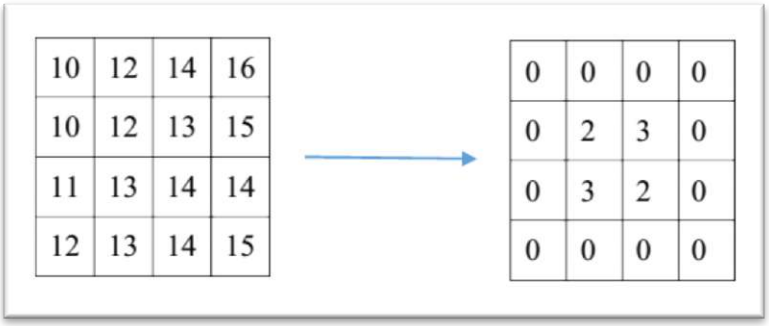


Figure 40. *Exemple de calcul de l'homogénéité.*

4. Conclusion

Nous avons présenté dans ce chapitre quelques algorithmes de détection de contour. Le partitionnement des blocs peut être considérablement simplifié pour des LCU homogènes (pas de contours). En premier temps, nous avons mis l'accent sur les détecteurs par gradient où chaque détecteur possède son propre masque de convolution. Afin d'optimiser le temps de calcul nous avons décidé d'utiliser un détecteur plus simple et qui ne présente aucune opération de multiplication (la multiplication est l'operateur le plus gourmand en temps de calcul), il s'agit d'un simple détecteur utilisant les différences entre un pixel et son voisinage. Les résultats d'implémentation font l'objet du chapitre suivant.

Chapitre 4 : Implémentation et Résultats

1. Introduction

L'objectif de ce travail est l'implémentation et l'optimisation de l'encodeur HEVC. Dans un premier temps, l'implémentation parallèle sur une plateforme multi-cœurs est présentée dans le but de réduire le temps d'encodage avec ou sans dégradation des performances de compression en termes de débit et PSNR. Si la plateforme est un simple monoprocesseur, l'optimisation est de type algorithmique pour éviter certaines formes de Cus lors du partitionnement, et dans ce cas le gain en temps de calcul est accompagné par une légère dégradation en PSNR et augmentation en débit. Les différents résultats d'implémentation sont présentés dans ce chapitre.

2. Outils et logiciels utilisés

Nous avons utilisé un ordinateur personnel portatif comme plateforme multi-cœurs, les caractéristiques de ce laptop sont résumées dans le tableau suivant.

Tableau 1. *Caractéristiques de la plateforme utilisée*

Caractéristiques	Valeurs
Processeur	Core i3 370M
Nombre de cœurs/threads disponibles	02/04
Fréquence du CPU	2.4 GHz
Taille de mémoire	06 GB
Version du système d'exploitation	Windows 7 64-bits

Pour le code source de l'encodeur HEVC, nous avons utilisé l'open source Kvazaar version 0.8.2. C'est un code libre pour l'usage académique sous la licence GNU GPLv2.1, et qui est téléchargeable gratuitement depuis le site https ://github.com/ultravideo/kvazaar.

L'avantage d'utiliser Kvazaar à la place du software de référence HM réside dans la possibilité d'exécution parallèle sur plusieurs cœurs et surtout sa rapidité due à l'optimisation par l'assembleur pour certaines fonctions. Il faut noter ici que nous avons proposé un nouvel outil de parallélisme au niveau frame par l'OpenMP. L'avantage de ce dernier est la réduction du temps d'exécution sans altérer les performances de compression. Les différentes implémentations sont développées sous Visual Studio 2015.

3. Implémentation et résultats

3.1.L'implémentation de base

Afin d'évaluer les performances de notre proposition il est nécessaire d'établir une référence pour la comparaison, c'est l'implémentation de base (sans optimisations). Un ensemble de séquences vidéo (Tableau 2) est utilisé pour nos tests, il s'agit d'une série de séquences de différentes résolutions recommandée par la norme HEVC.

Tableau 2. *Séquences vidéo utilisées*

Classes	Resolution	Sequences	Frame rate
Class D	416x240	***BasketballPass***	50
		RaceHorses	30
		BQSquare	60
		BlowingBubbles	50
Class C	832x480	***BasketballDrill***	50
		RaceHorses	30
		PartyScene	50
		BQMall	60
Class E	1280x720	***Vidyo1***	60
		Vidyo3	60
		Vidyo4	60
Class B	1920x1080	***ParkScene***	24
		Cactus	50
		Kimono1	24
		BQTerrace	60
		BasketBallDrive	50
Class A	2560x1600	***PeopleOnStreet***	30
		Traffic	30

Pour chaque séquence nous avons considéré quatre valeurs de QP (paramètre de quantification) qui sont : 22, 27, 32 et 37. Nous avons utilisé les paramètres du tableau 3 pour l'encodage de chaque séquence. Les résultats d'implémentation pour un seul thread (sans optimisation, sans parallélisme) sont présentés dans le Tableau 4.

Tableau 3. *Paramètres de configuration pour l'encodage*

Paramètres	Valeurs
GOPsize	1
Configuration	All Intra
Nombre de frames	100
LCU size	64x64
Max depthsearch	4

Tableau 4. *Résultats d'implémentation en utilisant 1 seul thread*

	Y-PSNR(dB)	Biterate (Kbps)	Temps d'encodage (s)
BasketballPass	37,60	5079,2441	12,91s
RaceHorses	36,04	8276,7714	15,11s
BQSquare	34,83	12531,4687	15,73s
BlowingBubbles	34,98	9606,0058	16,02s
BasketballDrill	36,77	19978,0468	51,97s
RaceHorses	36,43	29358,5703	56,70s
PartyScene	34,48	52084,3496	69,76s
BQMall	36,53	24447,0019	54,30s
Vidyo1	40,49	18215,6855	88,44s
Vidyo3	40,23	20017,5502	82,64s
Vidyo4	39,98	19636,2597	87,34s
ParkScene	36,87	104827,2813	265,20s
Cactus	36,49	98288,2548	260,74s
Kimono1	40,29	38720,8984	229,62s
BQTerrace	36,29	135896,1465	267,28s
BasketBallDrive	37,96	51882,1543	232,32s
PeopleOnStreet	38,04	179692,3789	502,47s
Traffic	38,36	172127,4023	508,03s

D'après le Tableau 4 on remarque que le temps d'encodage augmente lorsque la résolution augmente, ce qui confirme que le temps d'encodage est lié directement au nombre de LCU. Par contre et pour la même résolution, le temps d'encodage change (à titre d'exemple : de 12.91s à 16.02s pour la classe D, et de 51.97s à 69.76s pour la classe C) ce qui indique que le temps d'encodage dépond aussi des propriétés intrinsèques du contenu vidéo. On peut conclure que les séquences texturées nécessitent un temps d'encodage plus long par rapport aux séquences avec arrière-plan homogène.

3.2.Programmation multithreads

Pour réduire le temps d'exécution, nous proposons dans cette section d'utiliser une plateforme multi-cœurs, et par la suite affecter l'encodage d'une image à un thread séparé. Le nombre de threads disponible dans notre plateforme étant de 4, donc on peut encoder quatre images en parallèle, pour ce faire nous avons utilisé la programmation par OpenMP. Dans la fonction principale « main() » (située dans le fichier encmain.c) nous avons inséré la directive « #pragma omp parallel for » devant la boucle d'encodage.

La version initiale de la fonction « main » contient une boucle de type « while » dont la condition d'arrêt est la fin du fichier ou le nombre de trames, cette boucle est remplacée par

la boucle « **for (*cpt*=0 ;*cpt*<100 ;*cpt*++**) » pour permettre l'utilisation de la directive « #pragma omp parallel for » (où ***cpt*** est le compteur d'images).

Les résultats d'implémentation en utilisant deux et quatre threads sont donnés dans le Tableau 5. Pour la mesure des performances, nous avons calculé le gain en temps d'encodage par la formule :

$$\Delta T = \frac{Tproposé - Tinitial}{Tinitial} x100\ (\%)$$

Tableau 5. *Résultats d'implémentation en utilisant 2 et 4 Threads*

	ΔT (%) pour 2 Threads	ΔT(%) pour 4 Threads
BasketballPass	-44,66%	-54,98%
RaceHorses	-45,61%	-55,32%
BQSquare	-45,35%	-55,66%
BlowingBubbles	-44,99%	-55,77%
BasketballDrill	-45,92%	-55,66%
RaceHorses	-45,54%	-55,59%
PartyScene	-46,49%	-56,47%
BQMall	-45,07%	-56,20%
Vidyo1	-46,70%	-53,84%
Vidyo3	-45,55%	-54,12%
Vidyo4	-45,47%	-54,29%
ParkScene	-46,76%	-55,40%
Cactus	-47,07%	-55,72%
Kimono1	-46,64%	-54,94%
BQTerrace	-46,72%	-56,01%
BasketBallDrive	-46,26%	-56,61%
PeopleOnStreet	-45,48%	-54,05%
Traffic	-45,64%	-53,41%

Il est remarquable dans le tableau 5 que le gain en temps est dans le même ordre (autour de -46%) pour toutes les séquences quel que soit la résolution. Ce résultat est évident du moment que le traitement, en tous les cas, se divise sur les threads disponibles. Le passage de deux à quatre threads conduit à une amélioration en temps d'encodage, mais le gain ne se double pas (autour de -55%), car malgré l'exécution en parallèle il y a toujours des latences pour la synchronisation, puisque l'ordonnancement utilisé avec OpenMP est de type statique.

3.3.Anticipation du partitionnement selon l'homogénéité

Dans cette partie, nous proposons une terminaison anticipée pour le processus de partitionnement. En principe, le partitionnement se fait en quad-tree du niveau 0 (64x64) jusqu'au niveau 3 (8x8). Dans notre cas, le partitionnement peut être arrêté avant d'atteindre le niveau 3 si le bloc est homogène. En général, le partitionnement jusqu'à la taille 8x8 concerne les blocs avec plus de texture, et c'est le niveau 3 (8x8) qui est le plus pénalisant, puisque les blocs de prédiction peuvent aller jusqu'à la taille 4x4. Donc nous donnons plus d'intérêt pour éviter le niveau 3.

Au niveau de la fonction « `kvz_search_lcu` » du fichier « `search.c` », on effectue une analyse sur chaque bloc LCU à l'aide de l'algorithme d'homogénéité pour classifier le bloc en Homogène/Non-Homogène. Si le bloc est homogène, le maximum de la profondeur de recherche dans le quad-tree sera fixé à 2 au lieu de 4. L'organigramme de la Figure 41 résume les différentes étapes de traitement nécessaire.

Afin d'évaluer l'impact de notre proposition, nous utilisons les critères suivants :

1- Le gain en temps : $\Delta T = \frac{Tproposé - Tinitial}{Tinitial} x100$ (%)

2- L'augmentation en débit exprimé en (%) : $\Delta BR = \frac{BRprposé - BRinitial}{BRinital} x100$ (%)

3- La dégradation en qualité par : $\Delta PSNR = PSNR_{proposé} - PSNR_{initial}$

Les résultats d'implémentation sont donnés dans le Tableau 6.

D'après le Tableau 6, on remarque que le gain en temps d'encodage est variable d'une séquence à une autre (à la différence par rapport à l'exécution multithreads).Les séquences avec plus de blocs LCU homogènes présentent un gain plus grand que celles texturées.

L'optimisation du mécanisme de partitionnement dans l'encodeur HEVC est très importante dans le cas où la plateforme de type monoprocesseur, un gain de -17.83% à -35.30% est envisagé pour une faible dégradation de qualité (ΔPSNR) et faible augmentation de débit(ΔBR). Si la plateforme est multiprocesseur, on peut combiner les deux solutions précédentes pour avoir un gain important qui peut aller jusqu'à -69.55%

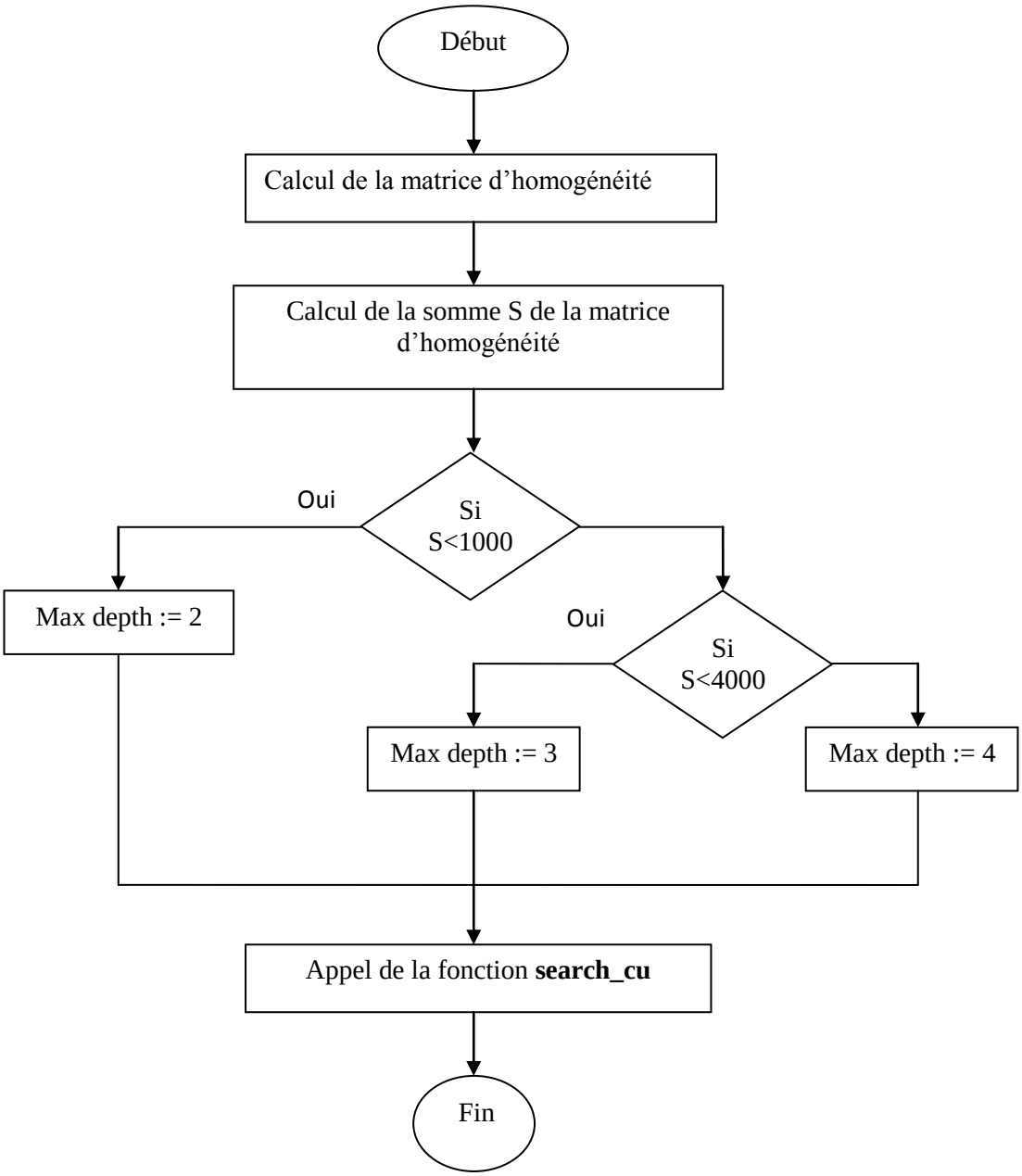


Figure 41. *Organigramme pour la sélection de la profondeur maximale de recherche*

Tableau 6. *Résultats d'implémentation par optimisation du partitionnement et multithreading*

	Optimisation du partitionnement			Optimisation du partitionnement & Parallélisme
	ΔT%	ΔBR%	ΔPSNR%	ΔT%
BasketballPass	-34,17%	3,84%	-0,11%	**-66,3%9**
RaceHorses	-33,52%	3,89%	-0,11%	**-68,96%**
BQSquare	-31,67%	10,38%	-0,37%	**-68,15%**
BlowingBubbles	-34,37%	3,18%	-0,13%	**-68,45%**
BasketballDrill	-16,33%	4,71%	-0,08%	**-65,83%**
RaceHorses	-31,79%	2,03%	-0,08%	**-67,94%**
PartyScene	-35,30%	4,31%	-0,14%	**-69,55%**
BQMall	-29,14%	3,77%	-0,16%	**-67,28%**
Vidyo1	-29,24%	4,08%	-0,09%	**-66,06%**
Vidyo3	-26,21%	5,55%	-0,14%	**-62,87%**
Vidyo4	-27,09%	3,33%	-0,06%	**-64,90%**
ParkScene	-28,62%	0,51%	-0,91%	**-68,39%**
Cactus	-25,17%	2,15%	-0,03%	**-65,59%**
Kimono1	-23,56%	-0,25%	-10,61%	**-66,66%**
BQTerrace	-25,11%	3,29%	-0,03%	**-64,51%**
BaskeBallDrive	-20,75%	1,30%	-0,01%	**-61,75%**
PeopleOnStreet	-28,76%	2,54%	-0,07%	**-65,78%**
Traffic	-17,83%	1,27%	-0,04%	**-66,19%**

4. Conclusion

Dans ce chapitre, nous avons présenté les résultats d'implémentation des différentes solutions proposées, dans le but de minimiser le temps d'encodage. L'implémentation sans optimisation (de base) est présentée en premier lieu pour constituer une référence pour la comparaison. Le gain apporté par l'implémentation multithread est homogène pour n'importe quelle séquence vidéo, ceci confirme le partage des ressources pour le traitement. Une autre solution est aussi proposée pour les plateformes monoprocesseur, cette dernière repose sur l'analyse des LCU pour anticiper le partitionnement.

Conclusion générale

La demande à des services vidéo de haute qualité est en croissance continue. Par conséquent, les normes de compression plus efficaces sont nécessaires pour surmonter l'énorme quantité de transmission de données, l'exigence de stockage de mémoire et la limitation en bande passante requise pour la transmission. HEVC est le dernier travail publié par JCT-VC pour remplacer le fameux H.264/AVC. En utilisant l'HEVC, le débit est réduit à la moitie pour la même qualité visuelle par rapport H.264, ce gain en performances de compression est accompagné par une complexité énorme et une grande consommation de ressources. D'ailleurs, l'exécution en temps réel reste limitée pour certains produits commerciaux.

Notre contribution dans ce champ de recherche concerne l'accélération de l'encodeur HEVC en proposants deux stratégies d'optimisation. La première solution était l'implémentation sur une plateforme multiprocesseur, de cette manière la charge de traitement se divise sur les différents cœurs de la plateforme, et par conséquent le temps de traitement se réduit. Les résultats d'implémentation montraient un gain (en temps d'encodage) d'environ-46% en utilisant 2 threads et -55% pour 4 threads et ceci quel que soit la résolution de la séquence vidéo. Le moyen utilisé pour le parallélisme était l'Open MP. Comme deuxième contribution, nous avons proposé d'anticiper le processus du partitionnement ce qui permet d'éviter la compression de certaines formes de blocs CU. Le gain apporté par cette dernière solution était de l'ordre de -17% à -35% selon la classe vidéo considérée. Mais il faut noter qu'une faible dégradation en PSNR et légère augmentation en débit sont constatées aussi. Finalement, les deux propositions peuvent être réunie en une seule solution en permettant un gain jusqu'à 69% en temps d'encodage. Seules les séquences de la classe D (416x240) qui peuvent être encodées en temps réel, les vidéo HD restent beaucoup loin de la vitesse 24frame/s.

Pour faire suite de notre travail, nous proposant comme perspective d'utiliser d'autres optimisations algorithmiques pour réduire la complexité d'encodage (à titre d'exemple, au niveau des 35 modes de prédictions intra). L'utilisation d'autres plateformes multi-cœurs plus 3performantes et avec plus de noyaux permet aussi de réduire le temps d'encodage.

Bibliographie

[1] : HARIZE Saliha, Etude et implémentation d'un codeur vidéo sur un système embarqué de type FPGA : Applications aux transformations orthogonales du codeur H264/AVC, 2013/2014.

[2] : KERFA Djoudi, Estimation de mouvement d'objets dans une séquence d'images, université des sciences et de la technologie d'Oran Mohamed Boudiaf, diplôme de Doctorat 2015/2016.

[3] : B. Pasquet, standards de codage védio, Béatrice Pesquet-Popescu, Département Traitement du Signal et des Images TELECOM ParisTech 15 mai 2012

[4]: Ahmed Mehaoua, cour Normes de Compression audio-vidéo, Université de Paris 5
UFR de Mathématiques & Informatique.

[5] :La compression video, Synthèse réalisée par Leslie HUIN – Master 2 IPM – 2013-2015.

[6] :By Mahsa T. Pourazad, Colin Doutre, Maryam Azimi, and PanosNasiopoulos, HEVC: The New Gold Standard for Video Compression, How does HEVC compare with H.264/AVC?JULY 2012.

[7] :Heiko Schwarz, Thomas Schierl, and DetlevMarpe, Chapter 3 Block Structures and Parallelism Features in HEVC, Novembre 2001.

[8] :AdriàArrufatBatalla, Multiple transforms for video coding, Signal and Image processing. INSA de Rennes, 2015. English.

[9] : Gary J.Sullivan Editors, High Efficiency Video Coding (HEVC), Algorithms and Architectures, 2014.

[10] : Gabriel Peyré. Le traitement numérique des images. CNRS. 2011.

[11] :Fuldseth A, Horowitz M, Xu S, Zhou M (2011) Tiles, Joint Collaborative Team on Video Coding (JCT-VC), Geneva, Mar. 2011.

[12] :Schierl T, George V, Henkel A, Marpe D (2012) Dependent slices, Joint Collaborative Team on Video Coding (JCT-VC), Geneva, Apr-May 2012.

[13] :Minhua Zhou1, Vivienne Sze, Madhukar Budagavi, Parallel Tools in HEVC for High-Throughput Processing, Texas Instruments Inc, 12500 TI Blvd., Dallas, TX-7524, 2014.

[14] Lewis (Bil) et Berg (Daniel J.). – Threads primer : a guide to multithreade dprogramming. – MountainView, CA, USA, SunSoftPress, 1996, 319p

[15] Flynn (Michael J.). – Some computer organizations and their effectiveness. IEEE Transactions on Computers, vol. C-21, n9, septembre 1972, pp. 948–960.

[16] T anenbaum (Andrew S.). – Modern Operating Systems. – Englewood Cliff, NJ, Prentice Hall, 1992.

[17] Dongarra (Jack J.), Meuer (Hans W .) et Strohmaier (Erich). – TOP500 supercomputer sites, 11th edition. In : Supercomputing 2000. – Dallas,USA, novembre 2000.

[18] IEEE. – IEEE Standard for Multithreaded Programming POSIX.1c. – IEEE Computer Society Press, 1995.

[19] Oaks (Scott) et Wong (Henry). – Java Threads. – Newton, MA, USA, O'Reilly& Associates, Inc., 1999, 319p.

[20] Dagum (Leonardo) et Menon (Ramesh). – OpenMP : An industry-standard API for shared-memory programming. IEEE Computational Science & Engineering, vol. 5, 1998, pp. 46–55.

[21] Sunderam (V . S.). – PVM : a framework for parallel distributed. Concurrency , practice and experience, vol. 2, n4, décembre 1990, pp. 315–339.

[22] MPI : A message-passing interface standard. International Journal of Supercomputer Applications and High Performance Computing, vol. 8, n3/4, 1994, pp. 159–416.

[23]Andréa Schwertner-Charao. Multiprogrammation parallèle générique des méthodes de décomposition de domaine. Modélisation et simulation. Institut National Polytechnique de Grenoble - INPG, 2001

[24] Christaller (M.). – Vers un Support d'Exécution Portable pour Applications Parallèles Irrégulières : Athapascan-0. – Thèse de doctorat, Université Joseph Fourier, Grenoble I, novembre 1996.

[25] Carissimi (Alexandre). – Le noyau exécutif Athapascan-0 et l'exploitation de la multiprogrammation légèr e sur les grappes de stations multiprocesseurs. – Thèse de doctorat en informatique, Institut National Polytechnique de Grenoble, France, novembre 1999.