SPIM

Thèse de Doctorat

THÈSE

En cotutelle avec l'Université de Monastir

Présentée par :

Wajdi ELHAMZI

Pour obtenir le

Grade de Docteur de l'Université de Bourgogne

Spécialité : Instrumentation et Informatique de l'Image

Définition et Implantation matérielle d'un estimateur de mouvement configurable pour la compression vidéo adaptative

Soutenue le 04 février 2013 devant le Jury :

Ridha BOUALLEGUE		Pr- Université de Carthage
Jean-François NEZAN	Rapporteur	Pr- INSA de Rennes
Mohamed ABID	Rapporteur	Pr- Université de Sfax
Johel MITERAN	Directeur	Pr- Université de Bourgogne
Rached TOURKI	Directeur	Pr- Université de Monastir
Julien DUBOIS	Co-encadrant	Mc- Université de Bourgogne

Table des Matières

Introduction

Contexte général

Grâce à l'évolution rapide des techniques de codage vidéo, accompagnée par les progrès technologiques dans le domaine du divertissement, des communications et de radiodiffusion, l'accès aux contenus vidéo numériques est en plein essor depuis plus d'une vingtaine d'années. La diversification des appareils capables de lire ces contenus (télévisions haute définition, tablettes, ordinateurs personnels, smartphones, etc) ainsi que la multiplication des supports de transmissions (réseaux wifi, réseaux cellulaires, fibre optique, 3G, 4G, etc) imposent aux diffuseurs d'investir du temps de développement et d'importants moyens financiers pour être en mesure de fournir une qualité de vidéo optimale quel que soit le contexte.

Face à ces enjeux importants, la nécessité de la compression des médias numériques et de la vidéo en particulier est évidente. Elle représente ainsi un levier majeur pour atteindre ces nombreux objectifs. En effet, la compression ou codage de la vidéo est le procédé permettant de réduire la quantité de données à transmettre ou à stocker, et donc de réduire les coûts qui y sont liés, ou de lever des verrous technologiques en apportant le signal vidéo là où on ne pouvait le transmettre auparavant, ou encore à un plus grand nombre de personnes simultanément.

L'amélioration du taux de compression demeure cependant un sujet de recherche crucial qui anime une forte communauté de laboratoires académiques et de compagnies internationales. Plusieurs standards de compressions ont ainsi été proposés afin d'aboutir à l'amélioration de l'efficacité de codage. Ainsi, MPEG-4 AVC/H.264 [Ric10][MPEG] permet de diminuer le débit nécessaire (jusqu'à 50 %) par rapport à son prédécesseur MPEG-2, ouvrant la voie à de nouveaux services, tels que la vidéo haute définition. Un nouveau standard de compression vidéo nommé HEVC (*High Efficiency Video Coding*), appelé H.265, est d'ailleurs en train d'être spécifié et permettra de répondre à une partie des défis à venir.

Un des éléments principaux des standards actuels de compression vidéo demeure l'estimation du mouvement [LKK+10]. Pendant cette phase, l'image courante est fractionnée en régions ou macroblocs, et l'on recherche dans les images précédentes où se trouvaient ces régions. C'est l'opération de mise en correspondance, dont on déduit, pour chaque macrobloc, un vecteur mouvement. Cette opération participe pour une grande partie à l'efficacité de la compression en permettant de supprimer les redondances temporelles. C'est aussi l'opération nécessitant le plus de puissance de calcul, puisqu'elle peut atteindre jusqu'à 70% de la charge de calcul d'un encodeur vidéo AVC/H.264. Pour autant la méthode de détermination des vecteurs de mouvement n'est pas normative : la stratégie de recherche de ces vecteurs est ouverte et reste le choix du concepteur. De ce fait, de nombreuses recherches ont été menées pour optimiser cette stratégie. De plus, dans les standards les plus récents, tels que H.264, différentes options ont été incorporées à la recherche initiale pour l'affiner : le fractionnement des macroblocs nommé « Variable Size Block » (VBS) et la recherche subpixélique du déplacement (typiquement 1/2, 1/4 ou 1/8 de pixel). L'étude bibliographique a permis de mettre en évidence d'une part la diversité des codages possibles et d'autre part l'impact de l'algorithme de recherche choisi et des raffinements sélectionnés sur les performances de codage (taux de compression, rapport signal sur bruit).

Cette thèse s'inscrit donc dans le contexte général de la compression vidéo et plus précisément dans la partie estimation du mouvement.

Objectifs et motivations

Au vue de la diversité des standards de codages, de l'hétérogénéité des média de communication (en particulier des bande-passantes associées) et des terminaux utilisés, il paraît primordial de pouvoir adapter les performances de codage en fonction des évolutions des contraintes liées à l'environnement, voire du contenu de l'image ou des événements détectés dans la scène.

D'autre part, les outils existants combinant rapidité, faible encombrement et faible consommation et qui réalisent la compression sont la plupart du temps basés sur des architectures spécifiques. En effet, devant la demande croissante en puissance de calcul des applications de traitement du signal et particulièrement, des applications de codage d'images fixes et vidéos, il est de plus en plus courant de faire appel à la puissance des architectures matérielles afin d'accélérer le temps de traitement et respecter les contraintes temps réel. Ce

type d'approche présente une puissance potentielle importante, mais la complexité de la description et le coût d'implantation d'une application peuvent s'avérer très élevés. Pour cette raison, plusieurs langages à haut niveau ont été mis au point afin de faciliter la conception hardware tout en profitant de la souplesse d'une description à un niveau d'abstraction le plus élevé possible.

Notre premier objectif était donc de proposer une architecture capable de concurrencer l'état de l'art, en termes de performances brutes d'encodage, et d'envisager plusieurs outils pour la description de cette architecture.

D'autre part, notre second objectif était que cette architecture soit flexible, permettant d'une part de modifier la stratégie de recherche et d'autre part de sélectionner les raffinements préconisés par les standards les plus récents. En effet, l'opération de mise en correspondance de blocs dépend de la stratégie choisie. L'étude exhaustive de toutes les positions du macrobloc dans la zone de recherche représente la technique de référence. Il s'agit aussi de la méthode la plus coûteuse en temps de calcul. D'autres stratégies, sub-optimales, utilisent un parcours particulier de la fenêtre de recherche pour limiter le nombre de tentatives de mises en correspondances, et sont de ce fait beaucoup plus rapides.

La flexibilité vise à pouvoir adapter et optimiser l'estimation du mouvement réalisée pour la prédiction de l'image courante par rapport à une image précédemment encodée/décodée selon le choix d'utilisateur ainsi que le type d'application. Il nous a donc été nécessaire d'identifier les éléments pouvant être utilisés au sein d'un modèle commun d'architecture afin d'arriver à notre objectif final, l'implantation de l'ensemble de l'architecture et la comparaison avec des architectures existantes.

Contribution

Les travaux réalisés au cours de cette thèse s'articulent autour de deux phases de recherche ayant pour but de proposer de nouvelles architectures matérielles d'estimateurs de mouvement flexibles et améliorant les performances en compression des codeurs vidéo actuels.

La première phase de nos travaux est ainsi davantage orientée vers l'étude générale algorithmique et architecturale des différentes stratégies de recherche afin de proposer une solution permettant de supporter les différents paramètres de configuration visant à améliorer

l'encodeur standard H.264/AVC de façon incrémentale. Dans cette phase, nous identifions principalement quatre contributions qui apportent des gains significatifs par rapport aux travaux antérieurs :

- Une analyse des différentes stratégies et architectures existantes, qui nous a amenés à définir un modèle modulaire et flexible.
- Une nouvelle architecture matérielle d'un estimateur de mouvement pixélique à taille de bloc variable, capable de supporter différentes stratégies.
- Deux nouvelles architectures des unités d'interpolation demi et quart de pixel, profitant de parallélisme des données, afin de réduire le temps de traitement de raffinement subpixélique.
- Deux nouvelles architectures concernant l'estimation de mouvement subpixélique (séquentielle et pipeline).

La seconde phase de nos travaux est davantage consacrée l'étude de l'optimisation de temps de mise sur le marché d'un tel système, par réduction du temps de développement. Une modélisation et description de type flot de donnée en langage CAL [EJ03] de l'estimateur de mouvement pixélique est réalisée. L'utilisation de la plateforme ORCC a permis de traduire cette description en langage VHDL. La comparaison d'une implantation basée sur le code généré par cette plateforme a pu alors être réalisée avec celle basée sur une description manuelle de l'architecture proposée. Une discussion est aussi menée pour démontrer l'apport de telles descriptions « haut niveau » au niveau de la conception d'une architecture hétérogène d'un estimateur du mouvement supportant la stratégie proposée dans cette thèse.

Ce travail a été effectué dans le cadre d'une coopération franco-tunisienne, projet CMCU intitulé « **Plateforme de compression vidéo pour des applications basse résolution incorporant des mesures temps réel de reconnaissance de formes »** au sein de deux laboratoires : Laboratoire Electronique, Informatique et Image (Le2i) de l'Université de Bourgogne et le Laboratoire d'Electronique et Microélectronique (LEME) de l'Université de Monastir.

De même nous avons contribué au projet « **Smart Camera** » du département Electronique du laboratoire Le2i. Ce projet consiste à proposer des méthodologies permettront de faciliter deux aspects fondamentaux de la conception de ce type de caméra. D'une part, il s'agit d'accélérer le développement de traitements d'images répondant aux contraintes fixées par le

capteur vidéo. D'autre part, afin de faciliter l'intégration de ces traitements au sein d'une Smart Camera, il s'agit de générer les interfaces externes requises par l'application et les lier automatiquement aux traitements.

Organisation du mémoire

Les aspects algorithmiques et architecturaux sont abordés dans cette thèse, afin de trouver une adéquation entre les besoins applicatifs, l'architecture et l'algorithme proposés. Le contexte applicatif, à savoir la nature des séquences, les performances désirées par l'utilisateur ainsi que la configuration choisie, sont considérés comme un élément de validation et d'évolution de la démarche méthodologique.

Comme le reflète le titre de la thèse, ce mémoire est composé de deux parties. La première, composée des chapitres 1 et 2, traite de la vidéo en vue de son codage avancé et d'une analyse pratique du standard de compression AVC/H.264, en insistant plus particulièrement sur la partie d'estimation de mouvement. La seconde, composée des chapitres 3 et 4, est dédiée aux architectures et implantations proposées de l'estimateur de mouvement configurable pour le codage de la vidéo H.264.

En détails, le manuscrit est organisé comme suit :

- Le premier chapitre est consacré à l'état de l'art des normes de compression vidéo. Durant cette présentation, les concepts de vidéo numérique et les traitements associés aux étapes d'encodage et de décodage de la norme H.264/AVC sont introduits.
- Le second chapitre présente les différents algorithmes ou stratégies de recherche et leurs architectures matérielles associées concernant l'estimation de mouvement pixélique et subpixélique, suite d'une étude analytique présentant les impacts du choix de la stratégie de recherche, la taille de blocs fixe ou variable ainsi que la précision utilisée, afin de proposer une architecture configurable selon les paramètres cités précédemment.
- Le troisième chapitre s'intéresse plus particulièrement aux architectures matérielles proposées d'un estimateur de mouvement configurable pour les deux phases : pixélique et subpixélique. Une comparaison avec des travaux récents à hautes performances et issus de la littérature montre l'efficacité et la pertinence des architectures proposées.

- Le quatrième et dernier chapitre est consacré à l'introduction d'une nouvelle approche de conception à haut niveau basée sur le langage flot de données CAL, permettant une génération automatique du code VHDL ou C associé afin de réduire le temps de mise sur le marché TTM.

Enfin, dans la conclusion générale, nous mettrons en évidence les avancées apportées dans la conception d'un estimateur de mouvement configurable compatible avec la norme de compression H.264/AVC ainsi que les perspectives concernant la conception d'architectures innovantes pour le profil haute définition et pour les nouvelles normes telles que la norme H.265 qui n'est pas encore standardisée.

Chapitre I : Compression et Codage Vidéo

1.1.Introduction

Le codage vidéo est le processus de compression et de décompression d'un signal vidéo numérique. Ce chapitre passe en revue la structure et les caractéristiques des images numériques et des signaux vidéo, et rappelle dans une introduction générale des concepts fondamentaux pour la suite de ce document, tels que formats d'échantillonnage et mesures de qualité.

En effet, la vidéo numérique est une représentation d'une scène échantillonnée spatialement et temporellement. Une scène est typiquement échantillonnée en un point dans le temps pour produire une image, et l'image elle-même est réalisée par échantillonnage spatial.

Ensuite, un bref aperçu est donné sur les standards de compression les plus courants, et nous terminerons par la présentation des spécificités du standard MPEG-4/AVC ou H.264.

Nous présenterons les modes de codages et les opérations d'estimation et de compensation de mouvement qui sont au cœur de notre préoccupation.

1.2. Introduction à la compression vidéo

D'une manière générale, une vidéo numérique non compressée se caractérise par son système de représentation des pixels, sa taille et sa fréquence de rafraîchissement (en images/s). Les formats varient selon l'application visée. Une telle vidéo non compressée est constituée des pixels de l'image brute, ce qui conduit à un besoin d'une bande passante importante (relativement aux autres objets multi-médias) pour transmettre ces données ou encore un besoin en mémoire très important pour assurer le stockage de toute l'information qu'elle contient. Cette quantité de données à stocker, à traiter, et éventuellement transmettre, pose un certain nombre de problèmes technologiques. De nombreuses méthodes ont été mises en place pour réduire ce volume de données, méthodes qui nécessitent de rappeler quelques généralités sur les images numériques, avant de détailler la compression vidéo proprement dite.

Une séquence vidéo numérique consiste en une suite d'images numériques. Chaque image est constituée d'une matrice de pixels. Ces derniers nécessitent une représentation permettant une description de ses composantes. En ce qui concerne les images à niveau de gris, les pixels sont codés la plupart du temps sur 8, 12 ou 16 bits. En ce qui concerne la couleur, les deux systèmes de représentation les plus fréquemment rencontrées dans les systèmes multimédia

sont [Ric10]: RVB (Rouge, Vert, Bleu) et YC_bC_r où Y est la luminance (niveaux de gris), Cb est la chrominance bleue et Cr est la chrominance rouge. L'espace couleur RVB est adapté pour la capture vidéo et l'affichage, tandis que l'espace de couleur YC_bC_r (appelé aussi YUV) est mieux adapté pour le stockage et la transmission. .

Dans l'espace RVB, les couleurs des pixels sont donc créées par la combinaison du rouge (R), vert (V) et bleu (B) dans des proportions variables. L'espace YC_bC_r se déduit de l'espace RVB par changement de repère.

Si l'espace de couleur YC_bC_r a la même résolution pour les trois composantes Y, C_b, C_r, le modèle d'échantillonnage associé est dénommé 4:4:4, comme présenté dans la figure 1.1, où chaque composante est codée de la même manière. Dans ce cas aucun sous-échantillonnage n'est réalisé, les trois composantes de chaque pixel étant utilisées.

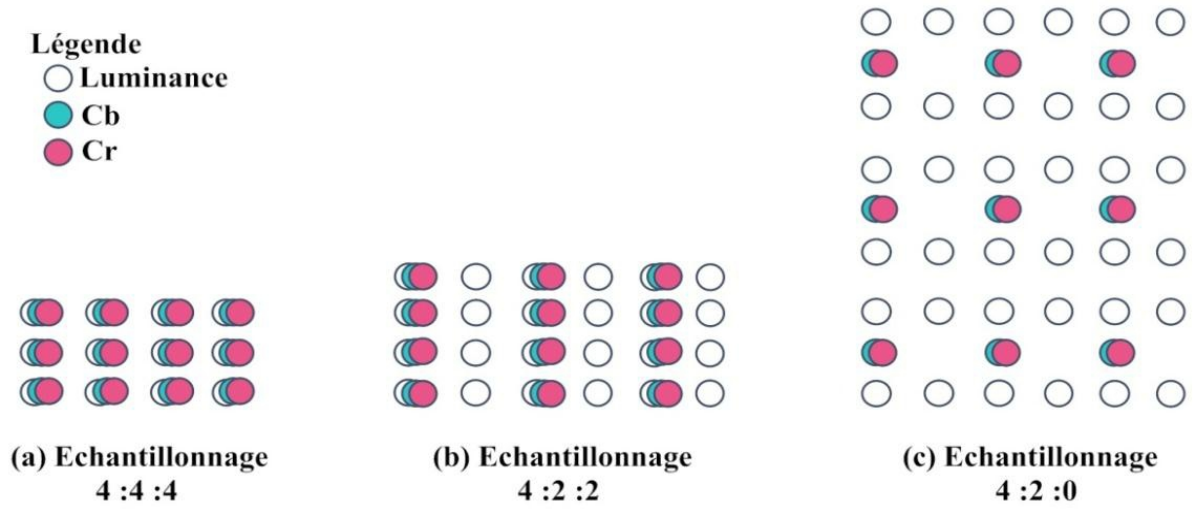


Figure 1. 1 : Représentation des modèles d'échantillonnage

Toutefois, l'œil humain est plus sensible à la luminance qu'aux couleurs. Il est donc possible d'échantillonner différemment les composantes, c'est-à-dire de réduire la définition des composantes de chrominance, sans perte significative de qualité visuelle.

Plusieurs formats ont ainsi été définis. Le modèle d'échantillonnage 4:2:2 (figure1.1-b)) divise par deux la résolution horizontale de Cb et Cr. Le modèle 4:2:0 (figure1.1-c)) consiste à réduire encore la résolution de Cb et Cr suivant les lignes par rapport au format 4:2:2. C'est le format le plus communément rencontré dans les normes de compression vidéo. Le modèle 4 :0 :0 consiste à ne conserver que la composante Y.

La vidéo numérique peut être considérée comme une séquence d'images fixes, affichées à une fréquence permettant de donner l'illusion d'un affichage continu. La fréquence minimum permettant cette perception continue de manière confortable pour l'être humain est de 25 images par seconde. Cette fréquence est donc souvent considérée comme un minimum pour les séquences vidéos. D'autre part, l'œil de l'être humain détecte un scintillement si le taux de rafraîchissement est inférieur ou égal à 50 Hz. Pour éviter ce problème, les écrans utilisent un balayage progressif ou entrelacé [Ric10] :

- Le balayage progressif (progressive scan) retrace toutes les lignes de la trame en un balayage.
- Le balayage entrelacé (interlaced scan) ou les lignes paires et impaires sont acquises et affichées de manière alternées. De fait, la fréquence de rafraîchissement correspond au double de la fréquence image.

Les écrans traditionnels à tube cathodique TV (CRT) affichent les signaux entrelacés, suivant les normes SDTV (NTSC, PAL, et SECAM). Cependant, les formats HDTV comme 720p et 1080p, affichés par les écrans modernes basés sur les technologies plasma, LCD et LED, sont progressifs. Actuellement, le balayage progressif est la technique dominante car elle double la résolution pour une même fréquence de rafraîchissement et évite les effets d'entrelacement.

L'évolution des capacités matérielles, des cartes graphiques et des écrans, a entrainé ces dernières années une évolution significative de la taille des images. Les applications actuelles multimédia utilisent généralement, pour les vidéos, des formats allant du Sub-QCIF (128×96 pixels) jusqu'à la haute définition QFHD (Quad Full High-Definition Resolution) (3840×2160 pixels). Dans un avenir proche, la taille de ces images va évoluer vers la Ultra Haute Définition Télévision UHDTV (7680 ×4320 pixels) afin de garantir une image de très haute qualité pour les utilisateurs.

Etant donné les tailles mentionnées ci-dessus, il est clair qu'une séquence vidéo brute (non compressée) nécessite une capacité de transfert très élevée. Les systèmes multimédia actuels (Smartphone, Tablette, TV ADSL, …) ne pourraient donc exister sans compression du signal faute de débit suffisant. A titre d'exemple, dans le cas d'une séquence de télévision HD (720p) de format 4:4:4, dont chaque composante YCbCr est représentée sur 8 bits avec une fréquence typique de 30 images par seconde, le débit nécessaire sans compression est de 30×(1280×720)×8×3= 663,55 Mbits/s tandis que le débit maximum pour une connexion à très

haut débit (fibre optique) est de l'ordre de 200 Mbits/s. Ce chiffre représente le débit maximum théorique abordable.

Une autre limitation de la vidéo non compressée est l'insuffisance des capacités des supports de stockage. Ainsi, le stockage d'une heure de vidéo au format HD (720p) de 3600s×663,55Mbit/s= 291Go. Un disque Digital Versatile Disc (DVD), d'une capacité de 4,7 Go ne permettrait de stocker que 16 minutes de format HD.

En résumé, les débits demandés par la vidéo non compressée sont ainsi beaucoup trop élevés par rapport aux bandes passantes disponibles pour leur diffusion, et les supports de stockage montrent rapidement leurs limites quant à la sauvegarde de données correspondantes. Face à ces limitation importantes et aux enjeux du multimédia, la nécessité de la compression des médias numériques et de la vidéo en particulier est évidente.

Afin d'assurer le succès des communications de contenus multimédias grâce à une bonne interopérabilité entre les acteurs des technologies de compression numérique, il a été nécessaire d'établir des normes [GKL01].

Ainsi, les travaux de normalisation internationale du codage vidéo est principalement coordonné par l'ISO/CEI « Moving Picture Experts Group » (MPEG) [MPEG], l'UIT-T « Video Coding Experts Group » (VCEG) [ITU], et la « Society of Motion Picture and Television Engineers » (SMPTE). Le groupe MPEG réalise notamment des recherches et développements algorithmiques qui permettent d'optimiser les taux de compression afin de garantir une utilisation optimale de la capacité de stockage ou de la bande passante. Les différents standards seront abordés dans le paragraphe 1.4.3 de la section 4. Ces optimisations visent en général à réduire la quantité d'information à transmettre ou à stocker tout en préservant au mieux le contenu visuel du signal [WOO05]. Pour cela, on cherche généralement à éliminer les redondances présentes dans le signal, grâce à un codage spécifique.

Nous allons maintenant donner un aperçu général du codage vidéo qui permet d'exploiter ces redondances. Nous décrirons pour cela d'abord les différents types de redondance, la manière de décomposer la vidéo en groupe d'images, nous rappellerons un des critères les plus utilisés pour comparer ces techniques, puis nous donnerons le schéma général d'un codeur vidéo.

1.3. Généralités concernant le codage

1.3.1. La redondance dans la vidéo

Les codecs, ou algorithmes de codage et décodage, ont donc pour but de réduire les redondances d'information dans les séquences pour mieux les transmettre ou les sauvegarder. Ces algorithmes n'exploitent pas tous les différents types de redondances de la même manière, ce qui conduit à différentes familles de codecs.

On distingue quatre types de redondance dans les signaux vidéo :

- ***La redondance spatiale*** *(codage Intra-image) :* ce type de redondance peut être exploité lorsqu'une image, prise indépendamment des autres, présente des zones uniformes plus ou moins grandes dans lesquelles les pixels sont similaires, voire identiques. La compression des images fixes (norme JPEG), utilise ce type de redondance.
- ***La redondance temporelle*** *(codage Inter-image*) : dans une séquence vidéo, la différence entre une image courante et la suivante est relativement faible, sauf lors d'un changement de plan. Cette similarité ou corrélation entre les images successives est la redondance temporelle. La technique de compression associée consiste à ne stocker que les informations qui sont modifiées lors du passage d'une image à une autre (codage Inter-image). Cette technique a eu un impact important dans l'évolution des normes de compression vidéo.
- ***La redondance statistique*** *(codage entropique)* : d'une manière générale, dans une séquence vidéo, les informations peuvent être classées selon certaines de leurs propriétés statistiques (occurrence, régularité, groupement, ...). Par conséquent, ces classes ne nécessitent pas d'être représentées avec un même nombre de bits. Ceci est utilisé dans le codage entropique.
- ***La redondance psycho-visuelle*** (*Transformation, Quantification*) : l'œil humain ne perçoit pas tous les types de détails de la même manière. Il est ainsi possible de coder différemment les hautes fréquences (détails, qui peuvent souvent être éliminés sans dégrader de manière significative les images) et les basses fréquences, qui sont très importantes pour la compréhension d'une image naturelle. Les techniques de transformées en cosinus discret (DCT) ou transformée en ondelettes (DWT), associée à une opération de quantification, sont utilisées pour exploiter ces types de propriétés.

1.3.2. Décomposition de la séquence vidéo

Dans un flux vidéo, toutes les images ne seront pas compressées de la même manière. Afin d'éviter les dérives qui pourraient être liées à une exploitation trop importante des redondances temporelles citées ci-dessus, il est nécessaire de transmettre régulièrement des images de référence notées I-Frame, donc le contenu ne dépend pas des images précédentes (son codage est dit INTRA). Entre deux images I-Frame, un certain nombre d'images codées différemment seront transmises : des images dites P-Frame (codage dit INTER) et des images dites B-Frame (codage bidirectionnel). L'ensemble de ces images, à partir de l'I-Frame, est noté Group Of Pictures, illustré figure 1.2 (GOP) [Ric10]. La fréquence des images I est variable ; elle donne la taille du GOP.

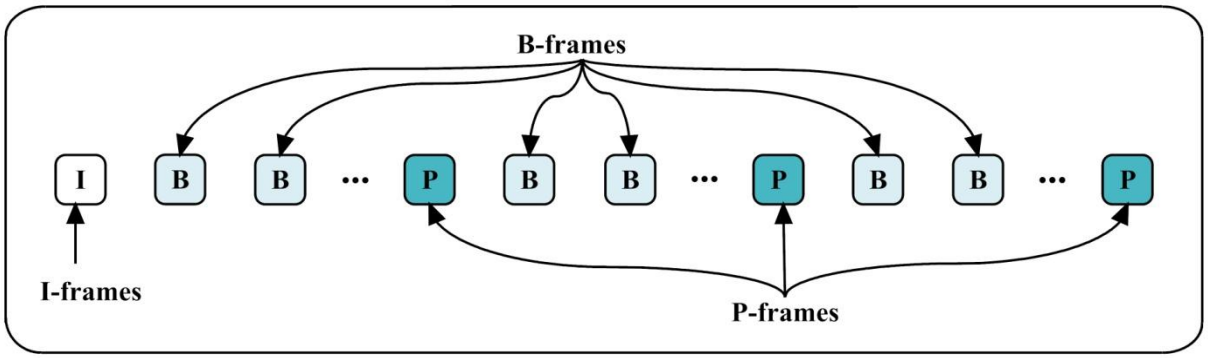


Figure 1. 2 : Structure d'un GOP (Group Of Pictures)

1.3.3. Critère de qualité d'image « PSNR »

Dans le but d'apprécier les performances de leur codeur, les concepteurs des codecs vidéo utilisent la métrique du rapport signal/bruit « PSNR » (Peak Signal to Noise Ratio) entre la séquence vidéo originale et sa reconstruction après la compression. Le PSNR entre une image originale ($\mathbf{F_R}$) et sa reconstruction ($\mathbf{F_C}$) est calculé comme suit :

$$PSNR_{db}(\mathbf{F_C}, \mathbf{F_R}) = \mathbf{10x\log_{10}}\left(\frac{\mathbf{255}^2}{\mathbf{MSE}(\mathbf{F_C}, \mathbf{F_R})}\right) \qquad (\mathbf{1.1})$$

Dans l'équation (1.1), le carré de la valeur maximale des luminosités disponibles, codée sur 8 bits, est divisé par l'erreur quadratique moyenne « MSE » (Mean Square Error) entre l'image originale et reconstruite. Cette erreur, pour des images de taille V×U pixels, MSE est définie de la manière suivante :

$$\mathbf{MSE}(\mathbf{F_C}, \mathbf{F_R}) = \frac{1}{(V-1)(U-1)} \sum_{i=0}^{V-1} \sum_{j=0}^{U-1} (\mathbf{F_C}(i,j) - \mathbf{F_R}(i,j))^2 \qquad (\mathbf{1.2})$$

Le PSNR est une mesure mathématique objective, qui produit des résultats indépendants de l'opinion de l'observateur, des conditions de visionnement (distance de lecture, éclairage, etc), et des technologies d'affichage. Cette métrique souffre d'un certain nombre de limitations : elle nécessite notamment une image originale irréprochable pour la comparaison, ce qui n'est pas toujours le cas.

Pour une image donnée ou une séquence d'images, un PSNR élevé indique généralement une bonne qualité de compression et un faible PSNR indique généralement une mauvaise qualité, ce qui permet d'évaluer et comparer les différents codecs et standards vidéo.

1.3.4. Schéma global du codeur vidéo

Le schéma générique d'une chaîne de codage est représenté figure 1.3. Il est constitué d'un encodeur et d'un décodeur.

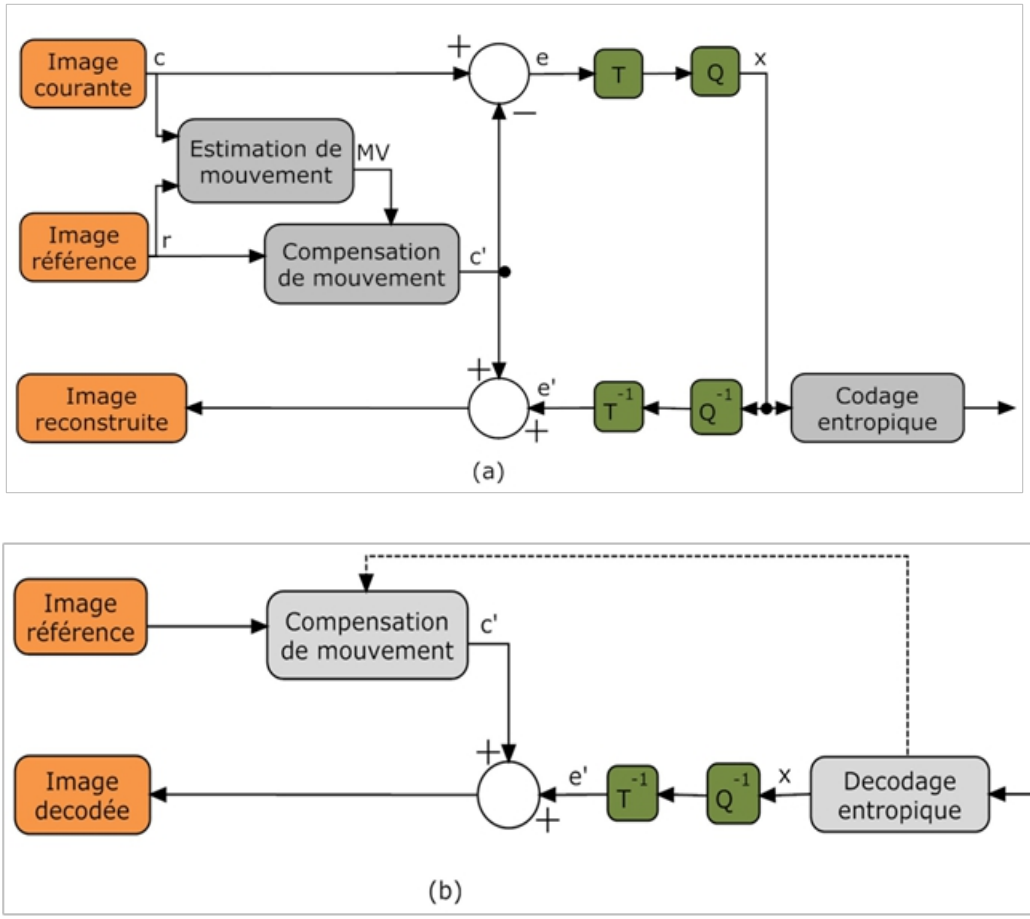


Figure 1. 3 : Modèle générique de codeur vidéo, (a) Encodeur, (b) Décodeur

L'encodeur construit une approximation des données d'entrée (les images de la vidéo non compressée), en cherchant à employer pour cela le moins de bits possibles, tout en conservant la meilleure qualité d'image possible.

L'encodeur vidéo se compose de trois unités fonctionnelles principales: un modèle de prédiction, un modèle d'encodage spatial et un codeur entropique [Ric10].

Le décodeur, de son côté, reçoit en entrée les données provenant de l'encodeur, et doit restituer en sortie une image aussi fidèle que possible à l'originale non compressée.

La plupart des standards de codage vidéo sont représentés par un modèle générique composé de blocs de traitement en cascade. Ce codeur intègre des fonctionnalités suivantes : la transformation (T), la quantification (Q), l'estimation de mouvement (ME), la compensation de mouvement (MC) et le codage entropique. Ces différentes fonctions seront décrites en détail dans le cadre de la présentation des normes MPEG4 / H 2.64, dans la section 5 de ce chapitre.

La première étape comprend des techniques d'estimation et de compensation mouvement (ME/MC) ainsi que la transformation, qui visent à exploiter la corrélation temporelle et spatiale de la séquence vidéo, afin d'optimiser les performances des étapes ultérieures que sont la quantification et le codage entropique. La technique la plus populaire utilisée pour l'estimation et la compensation de mouvement est basée une technique de mise en correspondance des blocs entre image courante et image de référence. En ce qui concerne la transformation qui exploite la redondance spatiale d'information, la technique la plus employée est la transformation en cosinus discrète (DCT).

Dans le décodeur, les traitements ci-dessus sont simplement inversés. Ces deux opérations sont, en elles-mêmes, complètement réversibles. Les pertes d'informations ne sont pas imputables à ces deux opérations mais à la quantification.

En ce qui concerne l'estimation du mouvement, à l'instant (t), le codeur reçoit une trame qui sera divisée en plusieurs blocs courants (c) de tailles égales. Par la suite, une comparaison avec les blocs de référence (r) est effectuée. La différence de position entre le bloc courant et le bloc estimé donne le vecteur de mouvement (MV) pour le bloc considéré. Ce dernier est envoyé vers le décodeur.

Les informations de mouvement sont utilisées pour calculer le bloc compensé (c'), qui est soustrait de la trame courante afin d'obtenir le bloc résiduel, qui représente l'erreur de prédiction (e). Le bloc transformé et quantifié (x) et les coefficients sont réarrangés pour le codage par entropie.

1.4.Les différents Standards de Compression

La complexité des applications multimédia et mobiles a constamment augmenté tout au long de la dernière décennie et les besoins en termes de compression des contenus vidéo ont évolué en même temps, ce qui explique l'existence d'une grande variété de standards dédiés à la compression vidéo. Les organismes ITU-T [MPEG] et ISO/MPEG [ITU] se partagent la normalisation de la plupart des standards de compression vidéo. Nous rappellerons pour mémoire la norme M-JPEG 2000 et la famille VP, mais notre travail s'est focalisé sur les normes de codage vidéo de l'ISO/CEI et de l'ITU-T et en particulier sur le standard publié conjointement par ces deux organismes : H.264/AVC (Advanced Video Coding) [Ric10]. La suite de ce chapitre sera consacrée essentiellement à une description globale de cette norme. Actuellement, les deux structures travaillent à l'élaboration d'une nouvelle norme commune (H.265/HEVC) [WOS+10].

1.4.1. Le M-JPEG 2000

Le Motion JPEG 2000 [ISO02a] est la partie 3 de la norme de compression d'images JPEG 2000 destinée aux applications vidéo. Son principe est d'appliquer un codage intra : un algorithme de compression de type JPEG 2000 (Joint Photographic Experts Groupe) est appliqué sur chaque image. Ce système autorise ainsi un accès aléatoire à n'importe quelle partie d'une vidéo car chaque image est traitée séparément. Cette norme est utilisée principalement dans les studios de montage numérique, qui nécessitent de se déplacer très fréquemment en avant ou en arrière dans le flux vidéo.

1.4.2. La famille VP

La compagnie **On2 Technologies** a développé plusieurs générations de codeurs vidéo libres VP (VP3, VP5, VP6, VP7 et VP8). En 2000, On2 a annoncé son premier codeur VP3 destiné à une utilisation en streaming. Un an plus tard, en 2001, une version VP3 Windows, puis son successeur VP4 ont été lancées. Des améliorations du codec ont été présentées dans les versions suivantes VP5, VP6 et VP7 entre 2002 et 2005.

Trois ans plus tard, la dernière version VP8 [DGL+11] a été annoncée. En mai 2010, dans le cadre du projet WebM [BWX11], VP8 a été racheté par Google. Actuellement, il est très utilisé dans le web ainsi qu'avec les systèmes de communication VoIP comme Skype et les nouveaux systèmes multimédias.

Ce type de codeur est caractérisé par une structure semblable à celle de H.264/AVC [Ric10] dans son profil de base, disposant des même outils classiques tels que la transformée 8x8, la prédiction bidirectionnelle la quantification et le filtrage adaptatives. Cette catégorie de codeur libre est devenue un vrai concurrent pour le codeur H.264, pour sa souplesse et la qualité offerte.

1.4.3. Les normes ISO/CEI et ITU-T

Les deux groupes ISO/CEI et ITU-T ont développé un grand nombre de normes internationales qui sont largement utilisées pour les codecs, pour le traitement et le codage audio/vidéo, notamment les familles MPEG et H.26x. Ces groupes ont également travaillé ensemble à la définition de la norme MPEG/AVC H.264 [Ric10] et son successeur HEVC/H.265 [WOS+10][ISO11].

Voici un bref rappel de l'historique concernant ces normes :

- **MPEG-1 :** En novembre 1991 le comité MPEG a lancé son premier standard [ISO91] permettant la compression des images en mouvements, en vue d'un stockage numérique d'une image ayant une résolution de 352 pixels par 240 pixels avec un débit pouvant atteindre 1.5Mb/s.
- **MPEG-2 :** Dans l'objectif de répondre aux limitations présentées par l'ancien standard MPEG1 (basse qualité d'image, résolution limitée), le standard MPEG2 [ISO94] a été annoncé en novembre 1994 supportant la compression progressive ou entrelacée de vidéos avec des débits variant de 1,5 à 100 Mb/s. Il a initialement été élaboré pour la transmission de la vidéo de qualité TV ayant un débit compris entre 4 et 9Mb/s. En outre, MPEG-2 est un schéma de compression couramment utilisée dans les DVD, ainsi que les émissions de radiodiffusion. Il est capable d'assurer le codage du SDTV et HDTV à des débits respectivement de 3,5 Mbit/s et 15Mbit/s.
- **MPEG-4 :** Dans le but d'englober les codecs MPEG existants et de leur rajouter une nouvelle dimension donnant naissance à un standard beaucoup plus souple et étendu, un nouveau standard MPEG-4 [ISO98] apparaît en 1995. Ce standard est destiné à la manipulation d'objets multimédias, et a comme objectif de coder de manière efficace

les séquences à très bas débit dédiées aux applications mobiles, mais il est capable de supporter des débits d'environ 1 Gbit/s.

Dans le même contexte, les normes de l'IUT-T : H.261 [ISO93], H.262 [ISO95] et H.263 [ISO96] ont principalement été développées pour la vidéo de communication par télécommunication et réseaux informatiques. Ces normes ont un grand nombre de caractéristiques communes avec les normes MPEG.

Voici un bref rappel concernant ces normes :

- **H.261 :** C'est une norme prévue pour la visiophonie (visioconférence) et pour les vidéos à base de Integrated Services Digital (ISDN). Elle est créée aussi pour soutenir des débits cibles de *m*x64 kbit/s (avec *m* compris entre 1 et 30).
- **H.262 :** Norme équivalente au format MPEG-2.
- **H.263 :** La norme de compression H.263 est conçue pour une transmission vidéo à très faible débit. L'inconvénient du faible débit est que l'image est dégradée lorsque les objets sont en mouvement. La norme H.263 [ISO96] était initialement destinée aux applications de vidéoconférence et non à la surveillance où les détails ont plus d'importance que la régularité du débit.
- **H.264 :** Le standard vidéo le plus attractif aujourd'hui est le H.264 [ISO02b] ou encore dénommé MPEG-4/AVC. Malgré sa grande complexité par rapport aux standards précédents, il est particulièrement apprécié pour ses performances en termes de compression. Nous nous focalisons dans la section suivante sur la description générale de cette norme.
- **Futur standard H.265 :** Durant la dernière décennie, le standard très répandu H.264/AVC a assuré les besoins applicatifs des systèmes multimédia, offrant des performances élevées par rapport aux anciens codeurs vidéo. Toutefois son efficacité s'est avérée insuffisante pour la compression ainsi que la transmission des nouvelles générations des résolutions (QFHD et au-delà). Par conséquent, le codec HEVC/H.265 [WOS+10][ISO11] s'apprête à prendre sa place, en multipliant par deux ses capacités de compression vidéo ainsi qu'en supportant les nouvelles résolutions. La finalisation de ce standard est prévue pour début 2013.

1.5.La norme MPEG-4/AVC H.264

1.5.1. Introduction

Le standard H.264 [Ric10][ISO02b] (appelé encore MPEG-4/AVC ou même MPEG-4 Part 10) est le plus répandu dans la compression vidéo. Il est dédié à une très large gamme d'applications pour la diffusion, le stockage, la conversation, et les services de streaming vidéo et a déjà été adopté pour la diffusion de la Télévision Haute Définition (TVHD). L'efficacité de la compression a augmenté de plus de 50% par rapport au format MPEG-2, permettant une meilleure qualité visuelle à un débit donné ou une bande passante réduite. Plusieurs comparaisons ont été réalisées avec d'autres codecs et les résultats ont toujours été en faveur de H.264. Cependant, à cet avantage vient s'ajouter l'inconvénient d'une grande complexité de calcul, faisant un défi considérable de l'objectif d'un encodage en temps réel ou encore du contrôle de la consommation en énergie des terminaux mobiles qui sont limités sur ce point mais aussi au niveau de leur puissance de traitement, de leur bande passante et de leur espace de stockage.

1.5.2. Structure de base

Le schéma du codeur H.264 [WSB+03] [LT03] est représenté figure 1.4 et figure 1.5. Nous y avons détaillé les opérations générales des codeurs vidéo déjà mentionnées dans la section 2.

Comme nous l'avons indiqué, la norme MPEG-4/AVC ou H.264 [OBL+04][STL04] permet un fort taux de compression au prix d'une complexité élevée. Toutefois, si la norme impose la méthode, les fabricants d'encodeurs restent libres au niveau des implantations de ces certaines parties du standard mais aussi certaines options. Des compromis sont donc fréquemment réalisés notamment sur l'estimation de mouvement, pour réduire la complexité de l'encodage qui pourrait atteindre jusqu'à 60% de la puissance de calcul.

Un encodeur vidéo H.264 réalise les processus de prédiction, transformation et encodage (Figure 1.4) afin de produire un bitstream conforme aux recommandations de la norme H.264. Un décodeur vidéo H.264 effectue les procédés de décodage complémentaires et de reconstruction de transformation inverse pour reconstruire la séquence vidéo originale.

La version décodée, en général, n'est pas identique à la séquence originale, car H.264 est un format de compression avec pertes, qui induit une dégradation de la qualité d'image.

Le schéma bloc modélisant la norme H. 264 est représenté figure 1.4 pour l'encodeur et figure 1.5 pour le décodeur. Le flux vidéo suit les graphes proposés selon deux chemins principaux.

Suivant le chemin de compression, une image courante est décomposée en plusieurs macro-blocs. Une image prédite P est obtenue en sélectionnant un des deux modèles de prédiction : soit le mode Inter par compensation de mouvement à partir d'une image de référence tout en passant par l'estimation de mouvement, soit, le mode Intra basé sur la prédiction spatiale réalisée à partir d'une zone courante déjà codée.

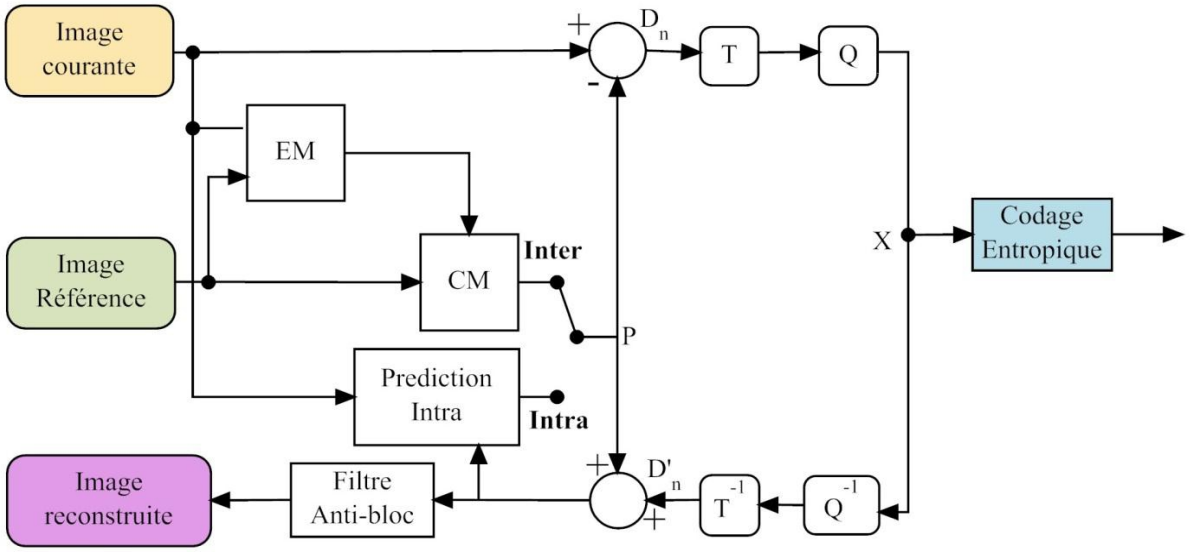


Figure 1.4 : Schéma bloc de d'encodeur H.264

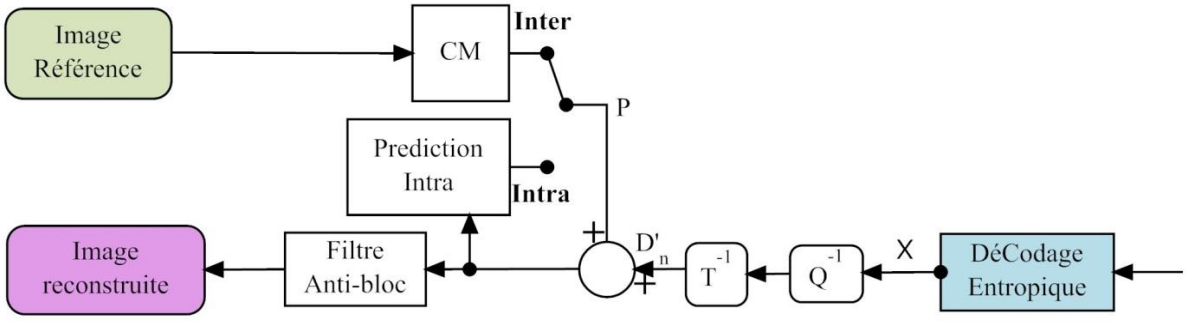


Figure 1.5 : Schéma bloc du décodeur H.264

L'image prédite résultante de la prédiction Intra ou Inter sera ensuite soustraite de l'image courante afin de produire un bloc résiduel (Dn), qui ensuite sera transformé puis quantifié afin de générer le flux de coefficients X. Ce dernier est transmis au bloc de codage entropique qui génère un flux binaire encodé. Ce même flux sera par la suite la source d'entrée du décodeur qui restituera la vidéo d'origine en répétant les mêmes étapes que pour le codage, mais inversement et symétriquement dans le temps.

1.5.3. Transformation

Cette étape de transformation (T) [Ric02], sert à réduire la redondance spatiale dans le même bloc en mettant en évidence les fréquences spatiales du signal. Pour cela, le standard H.264 utilise la transformée en cosinus discrète entière (DCT) ayant les mêmes propriétés que la DCT classique. L'avantage majeur de cette version de la DCT est la génération de coefficients entiers et non décimaux, ce qui permet à la fois d'éviter les problèmes des implémentations liés à la partie fractionnaire et de réduire le débit. Deux tailles de sous-fenêtres 4x4 et 8x8 sont utilisées dans le standard H.264 pour le calcul de cette transformation par DCT, ce qui permet de limiter la complexité de calcul.

1.5.4. Quantification

La quantification consiste à réduire l'espace des valeurs des coefficients issus de la transformation précédente, dans le but de réduire l'entropie de signal, afin de conserver l'information la plus importante pour l'œil humain [Ric02]. L'élimination des hautes fréquences est justifiée par le fait que l'œil humain est plus sensible aux erreurs de reconstruction liées aux basses fréquences spatiales qu'à celles liées aux hautes fréquences. La qualité d'une scène vidéo après quantification ne sera pas fortement dégradée si le débit binaire imposé n'est pas trop faible. Cette opération entraîne des pertes en information mais permet une compression des données importantes. Pour chaque norme de codage vidéo, il existe un ensemble défini de paramètres de quantification pour fournir les meilleures qualités de compression adaptées à différentes applications.

1.5.5. Filtre Anti-Bloc

Etant donné que la plupart des codeurs vidéo utilisent la compensation de mouvement, des artefacts visuels de type « blocs » indésirables peuvent être visibles lorsque la séquence est

reconstruite. Pour limiter cet effet, une étape supplémentaire introduite par H.264 est appliquée à tous les macro-blocs avant la reconstruction de l'image. Cette opération est le filtrage anti effet de bloc ou anti-bloc. Avant la reconstruction de l'image, ce processus de filtrage consiste à lisser les bords horizontaux et verticaux de tous les macro-blocs en utilisant des techniques de filtrage adaptatif. Par conséquent, la qualité d'image reconstruite objective (PSNR) et subjective (vision humaine) est améliorée [LJLB+03].

1.5.6. Codage entropique

L'opération de codage entropique est placée à la fin de la chaîne de codage vidéo. Le codeur entropique consiste à convertir le flux de symboles (luminance, chrominances) représentant les éléments de la séquence vidéo en un flux binaire compressé afin de le mettre à disposition du canal de transmission. Son principe est statistique : le codeur va attribuer des codes courts aux symboles les plus courants et des codes plus longs à ceux qui sont moins fréquents. Dans le standard H.264/AVC, les deux principaux codeurs entropiques qui ont été implémentés sont le CABAC (Context Adaptive Binary Arithmetic Coding) et le CAVLC (Context Adaptive Variable Length Coding) [WSB+03].

Ces méthodes sont basées sur un algorithme plus ancien, le codage à longueur variable « Variable Length Coding » (VLC). Toutefois, ce type de codage ne permet pas de s'adapter aux valeurs statistiques (fréquence d'apparition) des symboles traités au fur et à mesure du processus de codage. Le VLC n'est donc pas optimal, puisque l'efficacité de compression peut décroître en présence d'une structure d'image de plus en plus complexe. L'idée est donc de pouvoir s'adapter à un contexte, formé des valeurs des symboles précédemment codés, pour trouver le meilleur code du symbole courant. Le CAVLC et CABAC offrent ainsi la possibilité de compresser les images de manière plus efficace en ajustant leur action en fonction de chaque image, et ce, avec une grande précision.

1.6. Estimation et compensation de mouvement

1.6.1. Estimation du mouvement - généralités

Etant donné que nos travaux d'implantation matérielle ont concerné essentiellement la partie dite d'estimation du mouvement, nous allons maintenant détailler plus avant cette partie de l'encodage.

Dans une séquence vidéo, la différence entre une image et la suivante est relativement faible, sauf lors d'un changement de plan : c'est la redondance temporelle. Le but du codage inter-image ou compensation de mouvement est de prédire l'image à coder en fonction d'une image de référence et d'un champ de vecteurs de mouvement. Ensuite, le résidu est encodé avec les techniques présentées ci-dessus par la transformation et la quantification. De manière générale, le volume des données à transmettre (champ de vecteurs et résidus) est réduit par rapport à un codage intra-image, c'est-à-dire sans exploitation de la redondance temporelle.

Figure 1. 6 : Champs de vecteurs de mouvement et changement de plan

Une opération d'estimation de mouvement [ANR74][EDM+12][YSW89] est donc nécessaire afin de produire un champ de vecteurs le plus précis possible. Cette précision est indispensable car les performances d'un encodeur vidéo, notamment en termes de PSNR, dépendent beaucoup de la qualité des champs de vecteurs. Cependant, l'estimation de mouvement est une opération qui exige une puissance de calcul extrêmement importante, dès lors que la taille de l'image devient significative. L'estimation de mouvement peut représenter à elle seule de 60% à 70% de la puissance de calcul d'un encodeur vidéo H.264.

La figure 1.6 présente une suite temporelle d'images. Lorsque la caméra est fixe (image 1 et image 6) l'ensemble des vecteurs mouvements est représenté par le vecteur nul (points rouges) sauf pour les petites zones autour de la tête de la personne présente dans la scène. Cependant, si la séquence vidéo subit un changement (ici une translation de la caméra : image 2, image 3, image 4 et image 5), on remarque bien l'apparition d'un ensemble de vecteurs mouvements horizontaux, de direction opposée à celle de la caméra. Ce sont les vecteurs mouvements de chaque macro-bloc de taille 16x16.

Pour obtenir ces vecteurs, il a fallu réaliser une opération de mise en correspondance de blocs. En effet, un mouvement se traduit par un déplacement d'un groupe de pixels d'une image à l'autre. Il est donc nécessaire, pour chaque sous-fenêtre ou bloc d'une image, de rechercher son correspondant dans l'autre image. C'est l'opération dite de Block-Matching qui sera décrite en détails dans le paragraphe 1.6.3.

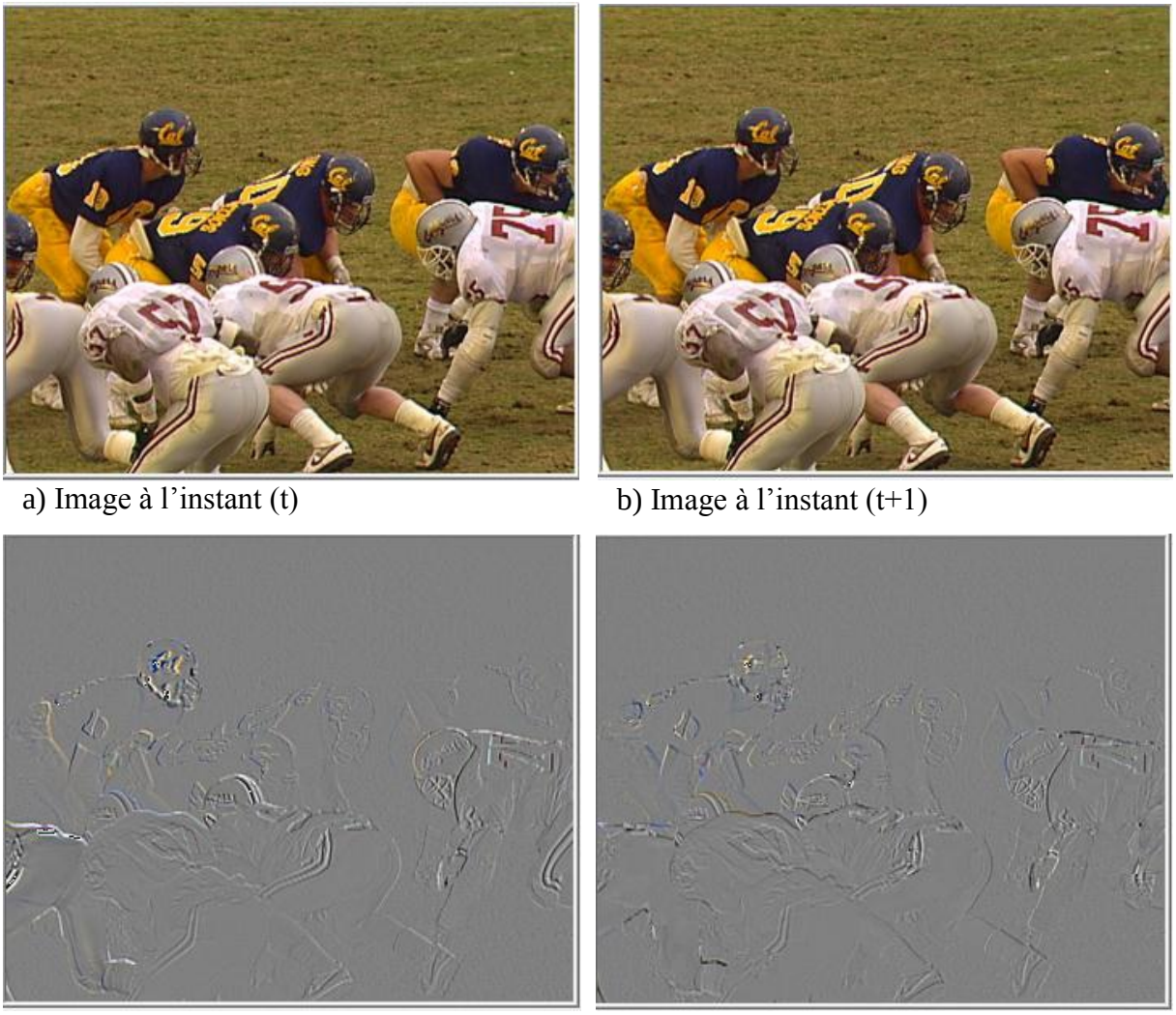
1.6.2. Compensation de mouvement

La plupart des méthodes de codage vidéo exploitent à la fois les redondances temporelles et spatiales pour atteindre la compression. Pour le domaine temporel, il y a généralement une forte corrélation ou similitude entre les images de vidéos capturées au même moment.

Les images temporellement adjacentes comme déjà montré dans la figure 1.7, à savoir les images successives dans l'ordre du temps, sont souvent fortement corrélées, surtout si le taux d'échantillonnage temporel est élevé. Dans ce contexte, la nécessité de tenir compte du mouvement est clairement exigée ce qui traduit l'importance de la compensation [HL92] de mouvement ou codage inter-image dans H.264 qui consiste à prédire l'image à coder à partir de l'image de référence et du champ de vecteurs estimé. Dans ce cas, le codage spatial de l'image en mode intra n'est plus nécessaire, il suffit de transmettre le résidu et les champs de

vecteurs de mouvement associés ce qui réduit considérablement le débit d'une séquence vidéo.

Usuellement, la compensation de mouvement peut être faite à partir d'une ou plusieurs images de référence afin d'améliorer la prédiction. En outre, chaque bloc peut être reconstruit à partir d'une seule référence ou d'une combinaison de deux références. On parle alors de bi-prédiction. La compensation de mouvement permet de réduire considérablement la bande passante nécessaire à la transmission d'une séquence vidéo.



a) Image à l'instant (t)

b) Image à l'instant (t+1)

c) Energie résiduelle sans compensation de mouvement

d) Energie résiduelle avec compensation de mouvement

Figure 1. 7 : Effet de la compensation de mouvement

1.6.3. Critère de mise en correspondance de blocs

Il existe différents types d'algorithmes de Block-Matching [NR80] pour estimer et donner des vecteurs de mouvements précis. Leur principe consiste à évaluer une distance entre le bloc de référence et tous les blocs candidats dans l'autre image, puis à choisir le bloc candidat minimisant cette distance.

Ainsi différentes distances ou critères d'erreurs sont communément utilisés pour ce type de calcul, comme la somme des différences absolues « Sum of Absolute Differences » (SAD, équation 1.3) ou la somme des différences des carrés « Sum of Square Difference » (SSD, équation 1.4) [Ric10]. N désigne la taille de bloc, CB le bloc courant, RB le bloc de référence, (i,j) le vecteur de mouvement (MV) et SR la plage de recherche :

$$SAD\ (i,j) = \sum_{m=0}^{N-1}\sum_{n=0}^{N-1}|CB(m,n) - RB(m+i,n+j)|,$$

$$SAD_{min} = min\big(SAD(i,j)\big), -SR \leq i,j < SR, \qquad (1,3)$$

$$SSD(i,j) = \sum_{m=0}^{N-1}\sum_{n=0}^{N-1}(CB(m,n) - RB(m+i,n+j))^2,$$

$$SSD_{min} = \min\big(SSD(i,j)\big), -SR \leq i,j < SR \qquad (1.4)$$

Le calcul de SSD (1.4) est plus complexe que celui de SAD (1.3). Pour un bloc de 16x16, l'équation de SSD nécessite par exemple 28 multiplications et 29 additions, alors que le calcul de SAD ne nécessite que 29 additions.

Il existe bien d'autres critères, comme la somme des Erreurs Quadratiques (SSE) (1.5) [Ric10], la Somme des Différences Transformées (SATD) (1.6) [Ric10], la Différence Moyenne Absolue « Mean Absolute Difference » (MAD), l'Erreur Absolue Moyenne « Mean Absolute Error » (MAE) etc.

$$SSE(i,j) = \sum_{m=0}^{N-1}\sum_{n=0}^{N-1}(CB(m,n) - RB(m+i,n+j))^2 \qquad (1.5)$$

$$SATD(d) = \sum_{m=0}^{N-1}\sum_{n=0}^{N-1}\left|H\left(CB(m,n) - RB(m+i,n+j)\right)\right| \qquad (1.6)$$

Où H est la transformé de Hadamard.

Nous nous sommes focalisés par la suite sur le critère SAD, car d'une part c'est le plus utilisé, et d'autre part sa complexité algorithmique et architecturale est réduite, puisque constituée uniquement d'additions de différences en valeur absolue.

1.6.4. Image de références multiples

Dans le but d'augmenter la probabilité de trouver une bonne correspondance entre blocs, plusieurs images de référence sont utilisées dans le codage H.264/AVC. Le nombre d'images utilisées peut atteindre cinq, permettant un codage plus efficace et plus performant. Cela signifie que le macro-bloc courant est comparé avec le nombre de références utilisées. Par conséquent avec deux images de référence, le nombre d'opérations nécessaires pour déterminer le meilleur macro-bloc correspondant sera doublé, avec quatre images ce nombre sera quadruplé. L'espace de recherche est donc élargi, ce qui a pour effet d'augmenter la complexité de l'estimateur de mouvement.

A titre d'exemple, l'image B peut être utilisée comme image de référence. L'utilisation de cinq images de référence pour la prédiction peut réduire le débit binaire jusqu'à 5~10% par rapport à l'utilisation d'une seule image de référence [WP03]. La figure 1.8 montre un exemple de multiples références.

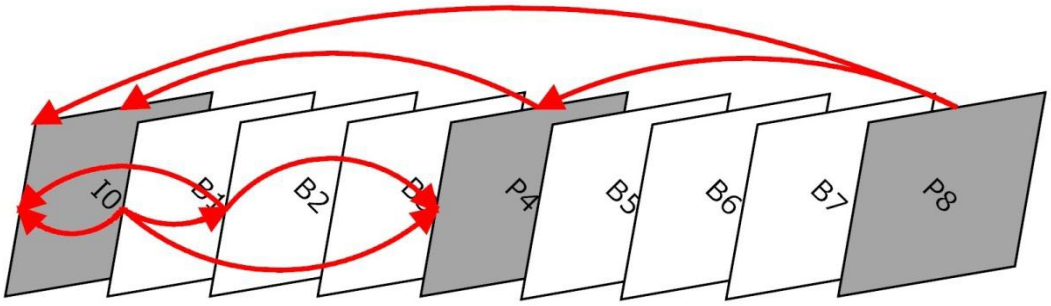


Figure 1. 8 : Exemple de références multiples

1.6.5. Taille de bloc fixe et variable

Dans les normes antérieures à H.264, les vecteurs mouvements sont en général estimés pour des sous-fenêtres ou macro-blocs de taille fixe (typiquement 16x16). Il s'agit de l'estimation « Fixed Block-Size Motion Estimation » (FBSME). Toutefois, ce découpage ne conduit pas à une définition très précise de l'estimation de mouvement. En effet, l'estimation FBSME demande le même effort pour estimer le mouvement des objets en mouvement et les

objets statiques. Cette inflexibilité provoque une faible performance de codage. Par ailleurs, deux objets en mouvement dans des directions différentes en un seul bloc peuvent aussi conduire à des estimations peu précises. La norme H.264/AVC [MGS+06], utilise donc un découpage permettant d'améliorer ces performances : il s'agit de l'estimation de mouvement à taille de bloc variable « Variable Block-Size Motion Estimation » (VBSME) [HJK+03] qui utilise de manière adaptative une plus petite taille de bloc pour estimer les petits objets en mouvement et une taille de bloc maximale (figée à 16x16 par la norme) pour les zones possédant un déplacement homogène au sein d'un macro-bloc, pour augmenter l'efficacité du codage.

Ainsi, chaque image d'une vidéo est toujours divisée en plusieurs macro-blocs de taille 16×16 pixels. Chaque macro-bloc peut être divisé en sous-blocs plus petits, de quatre manières: un bloc de 16×16, deux sous-blocs de 16×8, deux sous-blocs de 8×16, ou quatre sous-blocs de 8×8. Si le 8×8 soit le mode choisi, chacun des quatre blocs 8×8 peut être lui-même divisé encore en un bloc de 8×8, deux sous-blocs de 8×4, deux sous-blocs de 4×8, ou quatre sous-blocs 4×4 (Figure 1.9).

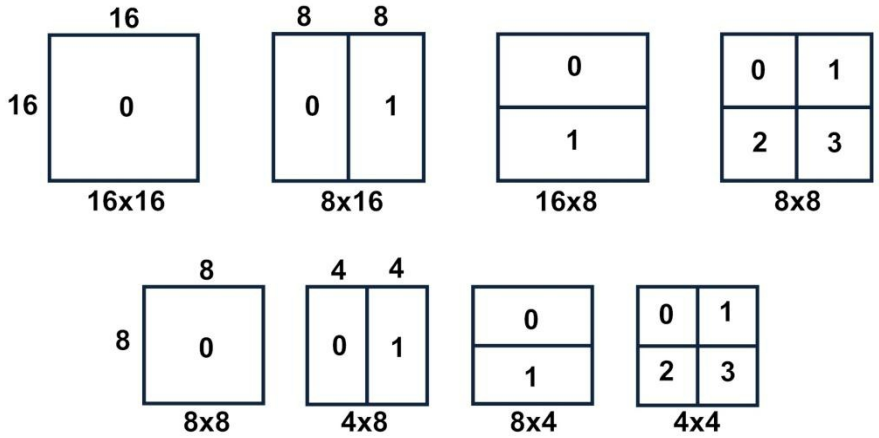


Figure 1.9 : Décomposition d'un macro-bloc

VBSME a été adoptée dans le codage vidéo dans les normes les plus récentes, y compris H.263, MPEG-4, et H.264/AVC. Par exemple, dans H.264/AVC, un MB avec une taille de bloc variable peut être divisé en sept types de blocs [SM04] comprenant : 4x4, 4x8, 8x4, 8x8, 8x16, 16x8 et 16x16. Bien que VBSME puisse permettre d'atteindre un taux de compression plus élevé, et il requiert d'une part une complexité de calcul plus importante que le FBSME, et d'autre part augmente la difficulté de mise en œuvre matérielle de l'estimation de mouvement [LL08] [CCH+06].

La compensation de mouvement de chaque bloc de 8×8 au lieu de chaque macro-bloc 16×16 réduit l'énergie résiduelle supplémentaire et la compensation de mouvement de chaque bloc 4×4 minimise l'énergie résiduelle. Par conséquent, les blocs de petites tailles peuvent produire de meilleurs résultats de compensation de mouvement. Toutefois, une taille de bloc plus petite conduit à une complexité accrue, avec plus d'opérations de recherche à effectuer et une augmentation du nombre de vecteurs de mouvement à transmettre.

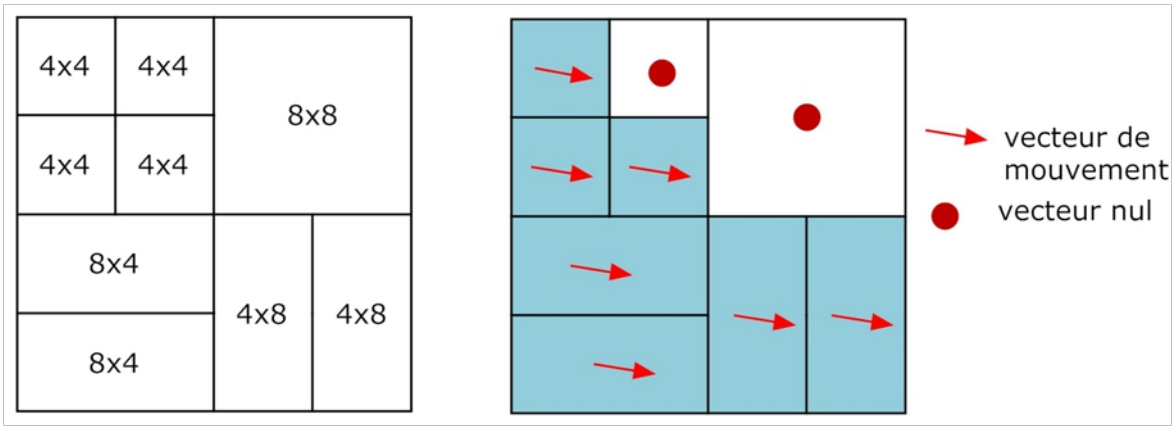


Figure 1. 10 : Partitionnement d'un macro-bloc

Pour un macro-bloc, l'augmentation du nombre de vecteurs de mouvement est coûteuse au niveau des ressources matérielles et du temps d'exécution. En contrepartie, la réduction au niveau d'énergie résiduelle améliore la qualité d'image codée. Un compromis efficace consiste à adapter la taille de bloc aux caractéristiques de l'image : les blocs de grande taille conviennent pour les zones homogènes (figure 1.10) de l'image alors que des blocs de petite taille favorisent les zones détaillées.

1.6.6. Estimation de mouvement subpixélique

Toujours en vue d'améliorer la précision de l'estimation des vecteurs mouvements, il est possible d'utiliser des techniques subpixel. Ainsi, l'estimation et la compensation de mouvement subpixélique [CHC04] assurent la recherche des positions subpixélique autour de la meilleure position pixélique. En outre, ce raffinement subpixélique permet de choisir la position qui donne le meilleur bloc et minimise l'énergie résiduelle.

La figure 1.11 illustre le concept d'estimation de mouvement au quart de pixel. Au cours d'une première étape, l'estimation de mouvement consiste à trouver la meilleure correspondance sur la grille des positions de pixels entières (cercles). La seconde phase

consiste alors à chercher les positions « demi-pixel » à proximité immédiate de cette meilleure correspondance (carrés sur la figure 1.11) pour voir si la précision peut être améliorée (meilleur critère de mise en correspondance). La troisième phase, si nécessaire, consiste cette fois à établir la recherche aux positions « quart de pixel » (triangles sur la figure 1.11). Le bloc ayant la position finale, (position entière, demi-pixel ou quart de pixel), sera soustrait du bloc courant afin de produire le résidu.

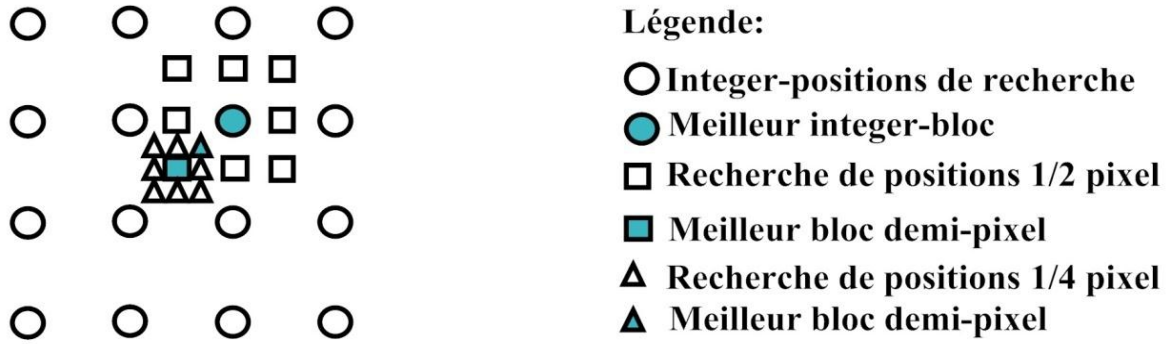


Figure 1. 11 : Positions entière, ½ pixel et ¼ pixel d'estimation de mouvement

Ainsi, une interpolation demi-pixel donne un gain significatif sur la compensation de mouvement pixélique comme nous le montrons dans le chapitre 2. Une interpolation quart de pixel permet une amélioration modérée par rapport au mode demi-pixel, et une interpolation au huitième pixel donnerait une amélioration encore plus légère, et ainsi de suite. L'estimation quart de pixel est couramment utilisée en raison de la précision obtenue, cependant son coût d'implantation est assez élevée et son accélération relativement difficile.

1.7.Conclusion

Dans ce premier chapitre, nous avons présenté les notions et les principes de base de la vidéo numérique. La première partie de ce chapitre nous a permis de rappeler les généralités de la vidéo numérique et l'historique des différents standards dédiés à la compression vidéo.

Dans la deuxième partie nous nous sommes intéressés plus précisément à la norme MPEG 4 AVC/H.264, en détaillant le fonctionnement de ses différentes phases. Ce standard fournit des performances nettement supérieures aux normes antérieures.

La dernière partie de ce chapitre introduit la phase d'estimation et de compensation de mouvement de la norme H.264/AVC d'une manière générale, et qui seront plus étudiés en détail étant donné son utilité au niveau des performances et sa complexité algorithmique et architecturale. Afin de résoudre les problèmes liés à cette complexité, les résultats de

nombreuses recherches pour diminuer la quantité de calcul durant la phase d'estimations de mouvement, tout en gardant le maximum de qualité, sont disponibles dans la littérature. Le chapitre suivant présente les algorithmes d'estimation de mouvement les plus représentatifs et leurs architectures matérielles.

Chapitre II : Algorithmes et architectures d'estimation de mouvement

2.1. Introduction

Comme nous l'avons vu au cours du chapitre précédent, l'estimation de mouvement demeure un des problèmes majeurs dans le domaine de la compression vidéo. Pour autant, cette opération s'avère essentielle dans la plupart des encodeurs vidéo. La qualité de l'estimation influe sensiblement sur les performances de la compression. Cette opération permet d'obtenir de forts taux de compression en exploitant les redondances temporelles observées entre plusieurs images consécutives d'une vidéo. Toutefois cette efficacité a un prix puisque ce traitement exige la plus grande partie du temps d'encodage [LKK+10].

Il existe de nombreuses variantes algorithmiques pour réaliser la phase d'estimation. Ces variantes sont essentiellement liées à la stratégie de recherche. Le choix de cette stratégie peut avoir un impact direct sur la qualité de l'encodage, le taux de compression, ainsi que sur le choix de l'implantation et par conséquent sur l'architecture en charge de son exécution.

Les implantations sont très variées : logicielles, matérielles ou mixtes, mais elles ne permettent pas toutes d'assurer un codage en temps réel du flux vidéo. En effet, les performances de codage à atteindre sont très changeantes. Elles dépendent du standard de compression sélectionné ainsi que de la configuration choisie (nommée aussi « profile ») mais aussi des contraintes imposées par l'application ou l'utilisateur (telles que résolution spatiale, fréquentielle, média de transmission, etc.). Bien sûr, les implantations matérielles ou mixtes sont les plus performantes en termes de vitesse d'encodage.

Pour faire face à la diversité des configurations d'encodage, il nous paraît primordial de définir une architecture matérielle ou mixte flexible, permettant le traitement en temps réel d'un flux vidéo en fonction de la qualité de codage. Afin d'obtenir cette flexibilité, nous proposons de nous focaliser sur la possibilité d'ajustement de la phase d'estimation du mouvement en particulier en fonction de la stratégie de recherche sélectionnée.

On trouve, dans la littérature, la description de nombreuses techniques et implantations relatives à l'estimation du mouvement. Ces dernières sont généralement classées en trois groupes : les méthodes basées sur l'utilisation de gradients, les méthodes fréquentielles et les méthodes de mise en correspondance des blocs. Parmi les centaines de propositions d'algorithmes et d'architectures, nous avons sélectionné les méthodes qui nous ont apparues comme les plus reconnues et les plus utilisées dans le contexte de la compression. De plus, dans ce chapitre, l'originalité, la fiabilité ainsi que la complexité de ces méthodes seront étudiées en vue d'une potentielle implantation (logicielle et/ou matérielle).

Dans la première partie de ce chapitre nous nous proposons de décrire les principaux algorithmes basés sur la technique de mise en correspondance des blocs. Les choix du concepteur au niveau de l'algorithme d'estimation du mouvement peuvent influer sur les performances d'encodage. Ainsi nous proposons une étude de l'impact sur l'encodage, de la stratégie de recherche utilisée, de l'utilisation de taille variable de blocs, et finalement du raffinement subpixélique. Les principales architectures matérielles associées à ces algorithmes seront ensuite décrites. Enfin, une présentation générale d'une architecture configurable est proposée comme une solution flexible et efficace pour l'estimation de mouvement.

2.2. Algorithmes et stratégies de recherche

2.2.1. Principe

Durant l'estimation de mouvement, des prétraitements sont réalisés. Chaque image est décomposée en plusieurs blocs de taille 8x8 ou 16x16. Ces blocs sont aussi nommés « macro-bloc ». Chaque bloc est considéré comme étant un élément indépendant. En effet, la méthode de mise en correspondance de blocs « Bloc-Matching » (figure 2.1) a pour objectif de rechercher la position la plus probable d'un macro-bloc extrait de l'image courante dans une image de référence passé ou future. La zone de recherche est généralement centrée dans l'image de référence sur la position du macro-bloc dans l'image courante. Sa taille est variable et fixée par l'utilisateur (typiquement 16 pixels). Par la suite, un balayage de la fenêtre de recherche est effectué jusqu'à la convergence de la recherche.

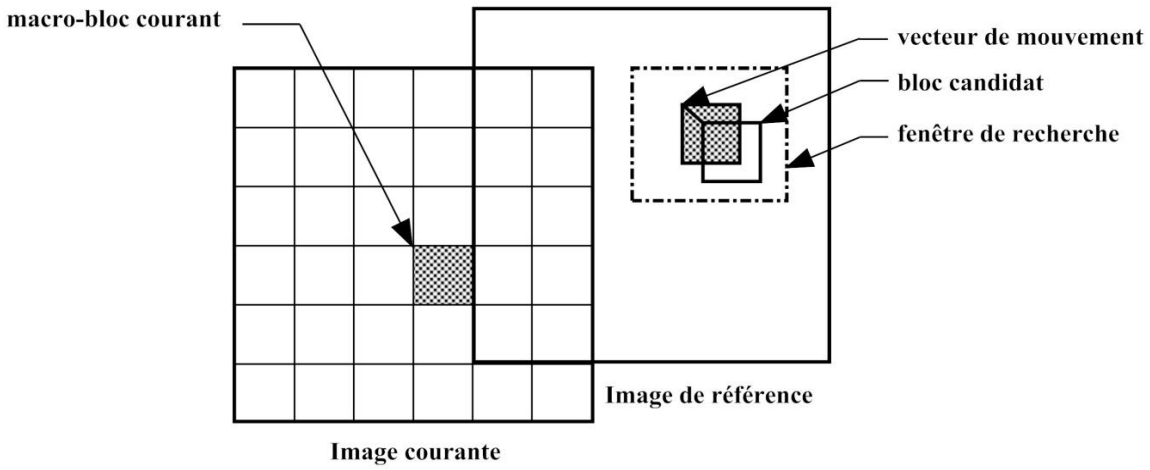


Figure 2. 1 : Mise en correspondance de blocs (1)

La convergence est définie comme la minimisation du critère de distorsion. Ainsi, pour chaque position du macro-bloc dans la zone de recherche une comparaison est réalisée entre les pixels des deux zones. Cette opération est nommée « matching ». Finalement, la position la plus probable correspond au maximum de similitude entre ces deux zones. Les positions relatives des deux blocs permettent d'obtenir directement le vecteur de mouvement (figures 2.1 et 2.2). Cette procédure est répétée jusqu'à ce que tous les macro-blocs de l'image aient été considérés. Un champ de vecteur est alors obtenu.

Dans ce contexte, plusieurs algorithmes et stratégies de recherche ont été proposés et développés dans le but d'assurer la phase de mise en correspondance des blocs. La grande diversité observée démontre bien l'importance majeure ainsi que la complexité algorithmique et architecturale de cette phase. Le choix de l'algorithme permet d'ajuster les performances de codage de manière à obtenir le meilleur compromis possible entre qualité d'image, taux de compression, temps de traitement.

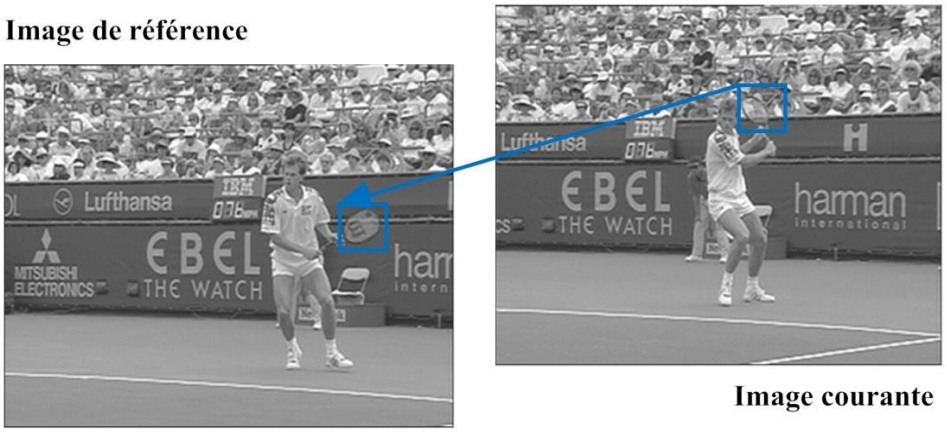


Figure 2. 2 : Mise en correspondance des blocs (2)

Dans la section suivante, une description générale ainsi qu'une classification ont été effectuée afin d'analyser et de déterminer les principales caractéristiques de chaque méthode.

2.2.2. Méthode de recherche exhaustive

L'algorithme de recherche exhaustive « Full Search » (FS) est un algorithme caractérisé par sa précision et sa régularité. En effet, son principe est de comparer au bloc de référence, tous les blocs possibles présents dans la fenêtre de recherche, et ce de manière exhaustive. L'algorithme 1.1 montre les étapes de la recherche exhaustive, où W et H représentent

respectivement la largeur et la hauteur du bloc, SR_H la plage de recherche horizontale et SR_V la plage de recherche verticale.

Pour comparer exhaustivement le macro-bloc à tous les blocs de référence dans la fenêtre de recherche, l'algorithme FS nécessite un très grand nombre d'opérations, car il réalise un balayage de toute la fenêtre de recherche suivant les directions horizontale et verticale. C'est l'algorithme le plus simple. Il donne les meilleurs résultats, mais c'est aussi le plus coûteux en termes de calculs. Par conséquent, de nombreuses techniques rapides ont été mises au point dans le but de réduire le nombre de « Matching » à réaliser, tout en donnant une solution la plus proche de l'optimum, telles que Three Step Search (TSS) [KIH+81], Cross Search (CS) [Gha90], Four Step Search (4SS) [PM96], Block-Based Gradient Descent Search (BBGDS) [LF96], Diamond Search (DS) [ZM00], HEXBS [ZLC02], Predictive Motion Vector Field Adaptive Search Technique (PMVFAST) [TAL01a], etc.

```
for w=0 to W/N do
   for h=0 to H/N do
      MV(w,h)=(0,0);
      SAD(w,h)=INIFINITE;
      for i=SR_H to SR_H -1 do
         for j=SR_V to SR_V -1 do
            SAD(i,j)=0;
            for x=0 to N-1 do
               for y=0 to N-1 do
                  SAD(i,j)+= |CB(x,y) - RB(i+x,j+y)|;
               endfor
            endfor
            if SAD(i,j) < SAD(w,h) then
              MV(w,h)=(i,j);
              SAD(w,h)=SAD(i,j);
            endif
         endfor
      endfor
      return with MV(w,h);
   endfor
endfor
```

Algorithme 1.1: Recherche exhaustive

2.2.3. Méthodes itératives

Les méthodes itératives font partie des méthodes rapides proposées dans la littérature. Le principe général de ce type d'algorithme est d'effectuer la recherche en plusieurs étapes, et plusieurs phases d'estimation. Chaque étape courante dépend de l'étape précédente. Le point ayant le minimum de distorsion sert à calculer les nouvelles adresses des points à tester de l'étape suivante. Le but étant de minimiser le nombre d'opérations de mise en correspondance de blocs, les stratégies consistent à optimiser un parcours parmi les blocs potentiels dans la fenêtre de recherche. Chaque algorithme suit pour cela une grille (ensemble de positions) de forme bien déterminée (carré, losange, cercle, hexagone) avec un rayonnement variable pour chaque étape. Les conditions de branchement entre les étapes et de convergence restent propres à chaque stratégie. Dans la suite, une description générale d'une variété des stratégies récursive est présentée.

2.2.3.1. L'algorithme Block-Based Gradient Descent Search

L'algorithme « Block-Based Gradient Descent Search » (BBGDS) [LF96] est un algorithme itératif qui consiste à évaluer le critère de matching pour chaque position d'une grille carrée de taille 3x3 pixels. Ainsi 9 positions sont testées à chaque étape. Si le minimum de critère de distorsion (MCD) est obtenu pour la position située au centre de la grille, l'algorithme est considéré comme convergeant et le processus de recherche est alors arrêté.

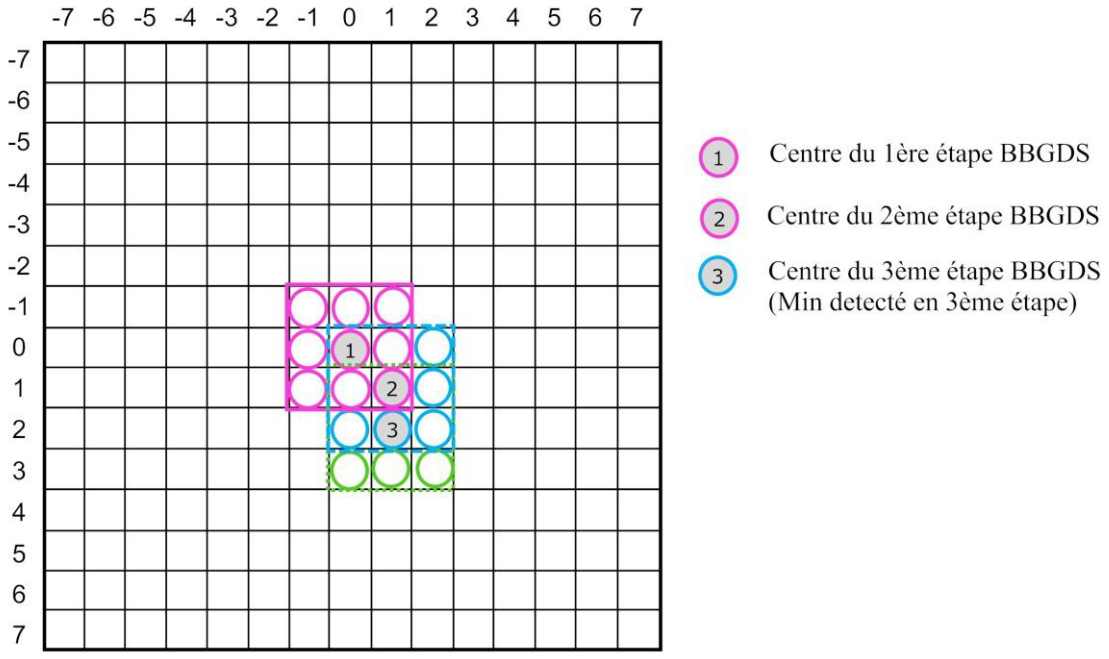


Figure 2. 3 : Exemple de calcul pour BBGDS

Sinon, la procédure de recherche se poursuit soit jusqu'à obtention d'un minima en position centrale soit en atteignant les limites de la fenêtre de recherche prédéfinie. Lors d'une nouvelle itération, la grille est définie autour du minima obtenu lors de la phase d'estimation précédente. Ainsi, une forte proportion des positions a déjà été estimée. Pour éviter de réaliser des calculs redondants et donc de ralentir le traitement, seuls 3 ou 5 points sont calculés, selon la position du point optimal.

La procédure de recherche de la méthode BBGDS assure que la recherche est effectuée dans la direction de descente du gradient. C'est la direction qui minimise localement la valeur de la distorsion. Le procédé est illustré dans la figure 2.3, où le vecteur de mouvement trouvé est (1, 2).

2.2.3.2. L'algorithme de recherche « Three Step Search »

L'algorithme de recherche « Three Step Search » (TSS) [KIH+81] permet de réduire la complexité des calculs en réduisant le nombre de points recherchés. La méthode TSS est recommandée par H.261 et MPEG en raison de sa simplicité et son efficacité. La figure 2.4 illustre les étapes de recherche de cet algorithme. Le nombre encerclé représente l'ordre de l'étape de recherche. Les cercles de fond gris indiquent le choix final de chaque étape.

Dans la première étape, les neufs premiers candidats autour du centre (x,y). Deux positions voisines dans le motif sont distantes d'un pas noté "d". Ainsi les positions des neufs points considérés sont alors (x-d, y-d), (x, y- d), (x + d, y- d), (x-d, y), (x, y), (x + d, y), (x- d, y + d), (x, y + d), et (x + d, y +d). La position permettant d'obtenir le minimum de distorsion (SAD, SSD, MAE, MAD, …) sera le centre de la prochaine étape. Dans la deuxième étape, la recherche s'effectue autour du minimum obtenu lors de l'étape précédente, mais avec une réduction de moitié du pas d. Le processus est réitéré lors de la troisième étape et neuf positions sont ainsi considérées. Lors de cette troisième et dernière phase la distance d est à nouveau réduite d'un facteur deux, les neuf positions se trouvent ainsi voisines. Le point associé au minimum d'erreur est considéré comme le déplacement et permet d'établir le vecteur de mouvement associé.

La figure 2.4 illustre cette recherche en 3 phases. Nous limitons la plage de déplacement ±7x7 c'est-à-dire entre (-7, 7) aussi bien pour les abscisses que pour les ordonnées. A la fin de la recherche, l'algorithme converge sur la position (7,-6). Pour cette zone de recherche, l'algorithme de recherche exhaustive FS teste 15x15=225 matching, néanmoins, TSS teste seulement 9+8+8= 25 matching.

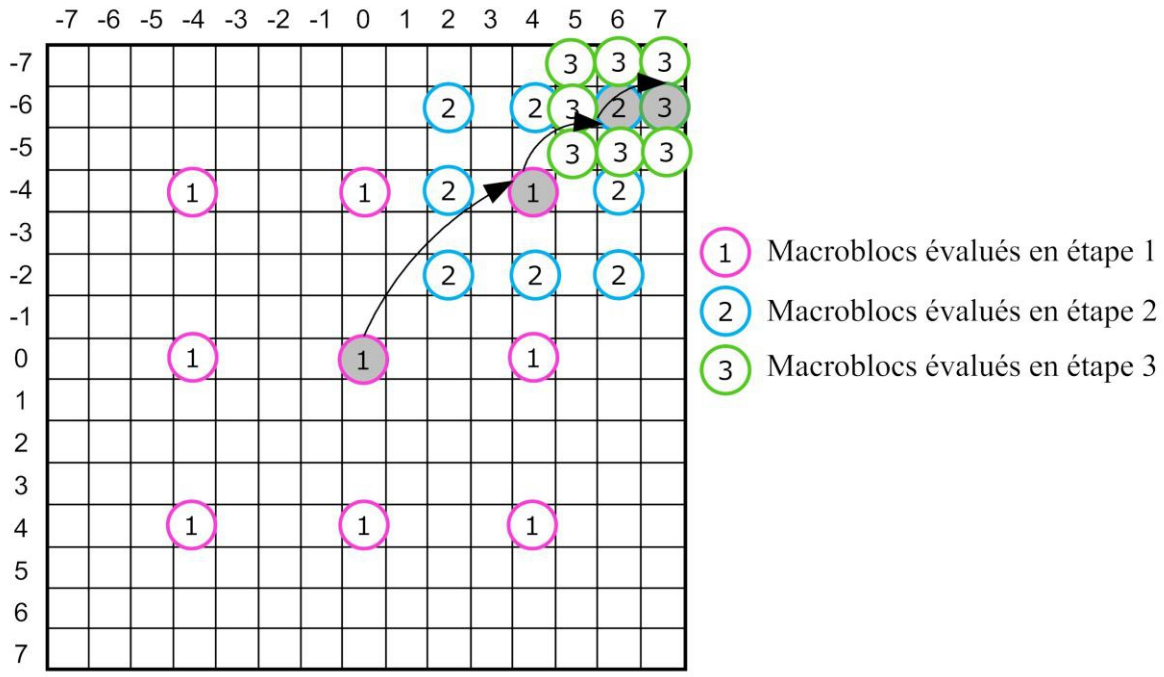


Figure 2. 4 : Algorithme Three step search (TSS)

Le nombre des points à tester est ici divisé par 9 par rapport à la méthode de recherche exhaustive, ce qui représente une accélération importante en temps de traitement. Le nombre de matching est fixe pour une recherche en 3 phases mais la zone de recherche doit être limitée pour conserver une bonne qualité de détection (et donc de codage !). Pour autant, même avec une plage de recherche limitée, l'algorithme n'assure pas une convergence vers le même minima de distorsion que la méthode FS. Ainsi l'algorithme TSS dégrade généralement la qualité d'images, surtout dans les séquences de mouvement rapides. Pour remédier à ce problème, l'algorithme d'estimation de mouvement New Three Step Search (NTSS) [LZL94] a été proposé comme solution. Le NTSS se distingue par rapport à TSS par l'ajout de huit nouveaux points pendant la première étape autour du point central, en plus de conditions d'arrêt supplémentaires. L'algorithme NTSS conserve la simplicité et la régularité de la méthode origine TSS, il fonctionne mieux que TSS en termes de robustesse et d'erreur de compensation de mouvement, et est tout à fait compatible avec TSS en termes de la complexité de calculs.

2.2.3.3. Recherche à quatre étapes

L'algorithme de recherche en quatre étapes « Four Step Search » (4SS) a été proposé par LM Po et WC Ma [PM96] en 1996 comme étant un algorithme rapide et inspiré de

l'algorithme TSS. Cette méthode utilise comme pour l'algorithme TSS un modèle de recherche uniforme en grille carré. Les deux algorithmes TSS et 4SS se ressemblent, en utilisant le même motif de recherche en carré. Cependant, il existe deux points de différences majeures entre ces deux algorithmes. Premièrement, dans le cas de l'algorithme 4SS, la taille de motif est figée pour les premières étapes, et permet contrairement à la méthode TSS d'obtenir des points de chevauchement entre celles-ci, et donc de réduire le nombre des points testés. Deuxièmement, le 4SS offre la possibilité d'un arrêt conditionnel permettant de réduire le nombre total de points cherchés.

La figure 2.5 montre une description générale de fonctionnement du 4SS. A la première étape, une grille en carré de taille 5x5 pixels est sélectionnée comme étant le motif de base de recherche. Elle comporte les neufs points candidats. Cette grille est appliquée pour les trois premières étapes, cependant pour la quatrième étape cette grille est réduite pour une taille de 3x3 pixels. Le candidat ayant obtenu le minimum de critère de distorsion (SAD, SSD, MAE,…) sera le centre de l'étape de recherche suivante (point du fond gris sur la figure). Afin d'optimiser les nombres des points recherchés, une technique d'arrêt conditionnelle est utilisée par le 4SS. Si le centre de la zone de recherche obtient le minimum de distorsion, l'étape de recherche suivante est directement la grille de taille 3x3 de sorte que le nombre de points de recherches requises peut se réduire, dans le meilleur des scénarios à 17 matching (9+8=17).

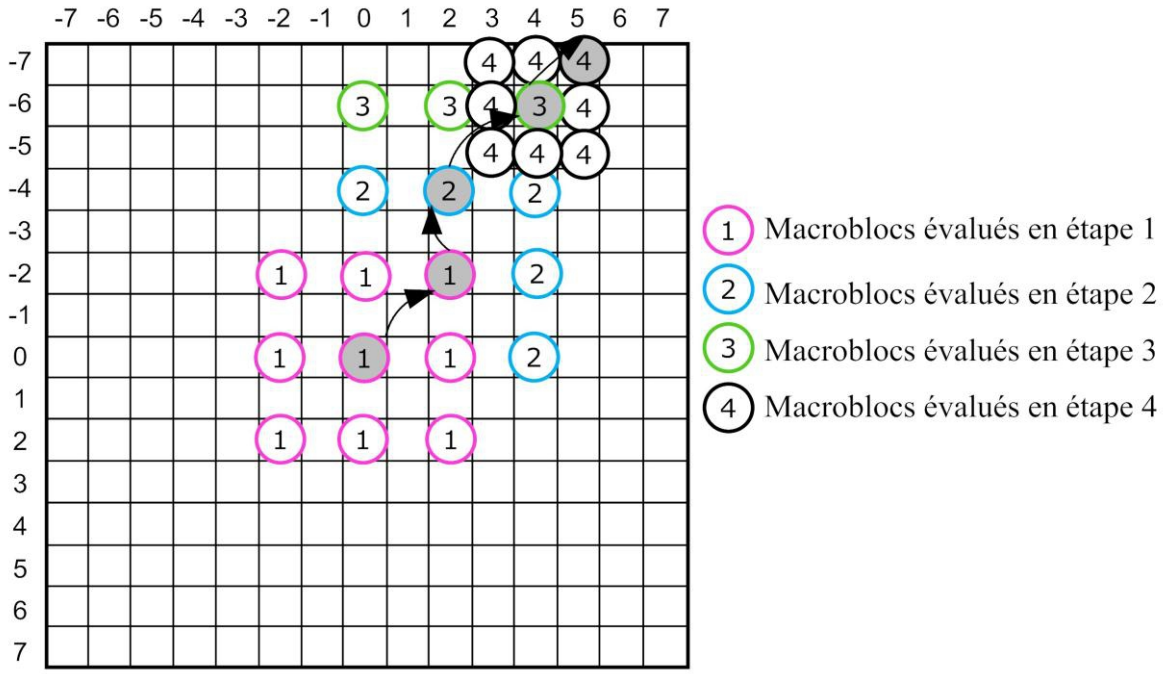


Figure 2. 5 : Algorithme de recherche à quatre étapes (4SS)

En outre, à chaque étape, il existe plusieurs points de recouvrement. Selon la position du minimum, c'est seulement 3 ou 5 points qui demeurent à calculer, le reste des points a déjà été estimé. Ainsi, lors des 4 phases consécutives d'estimation, le nombre maximum de positions considérées sera donc égal à 27 (9+5+5+8). Le nombre de points varie donc entre un minimum de 17 et un maximum de 27. Bien que dépendant de la nature de l'image, le nombre de points considérés se retrouve en moyenne généralement inférieur à celui traité avec la méthode TSS. Les performances de codage peuvent être légèrement améliorées tel que l'attestent les travaux de Ismail et al [IMS+09]. Ainsi le PSNR peut s'accroitre de 0.4db.

2.2.3.4. Recherche en diamant

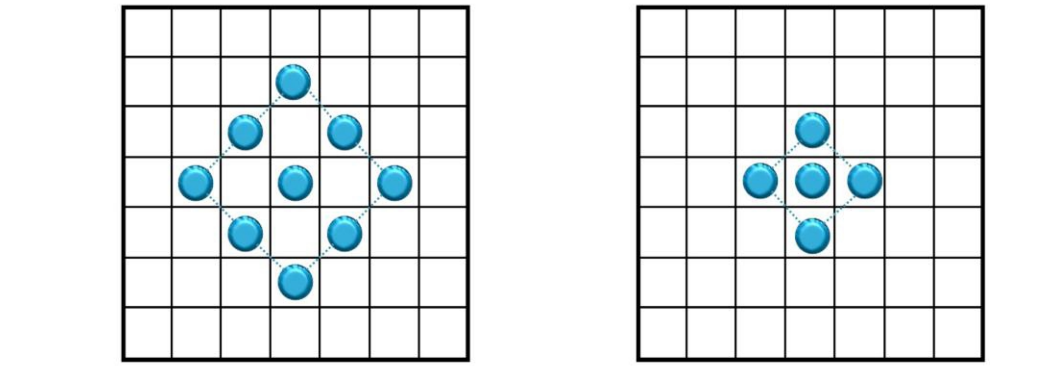
Fondamentalement parlant, la forme et la taille du modèle de recherche exploitée dans les algorithmes rapides détermine non seulement sa vitesse de recherche mais influe également sur les performances d'estimation du mouvement. Généralement, de multiples points de minima locaux existent dans la fenêtre de recherche, en particulier, pour les séquences d'images présentant une forte densité de mouvements différents. Par conséquent, en utilisant une grille de recherche de petite taille et un pas d séparant deux positions voisines égal à 1, (tel que dans la méthode BBGDS), il est tout à fait possible d'être piégé dans un minimum local pour les séquences vidéo où existent des mouvements importants.

Lorsque la distance d s'accroît, comme notamment avec l'algorithme TSS, la méthode de détection devient plus robuste pour la détection de mouvements rapides mais il est toujours possible de tomber dans un minimum local.

Un nouvel algorithme de recherche suivant une grille en diamant « Diamond Search » (DS) [ZM00][IMSB09] a été développé pour limiter cet inconvénient.

La méthode DS est basée sur neuf points de recherche initiaux, mais ces points de recherche forment un diamant au lieu d'un carré, induisant une distance d légèrement supérieure à 1, qui permet d'éviter d'évaluer les distorsions de manière exhaustive et introduit des déplacements diagonaux de la grille. D'autre part, la méthode utilise deux grilles de recherche de tailles différentes, toutes deux en forme de diamant, comme illustré dans la Figure 2.6.

Le premier motif, appelé « Large Diamond Search Pattern » (LDSP), comprend neuf points d'estimation : un point central et huit points formant un losange. Le deuxième motif, est composé de cinq points d'estimation, et possède la forme d'un petit diamant, « Small Diamond Search Pattern » (SDSP).



Grille de recherche en diamant large (LDSP)

Grille de recherche en diamant réduit (SDSP)

Figure 2. 6 : Modèle de recherche utilisé par DS

Dans la procédure de recherche de l'algorithme DS, le LDSP est utilisé sur plusieurs itérations jusqu'à ce que le minimum de critère de distorsion soit obtenu au point central. La grille de recherche est alors changée de LDSP à SDSP qui représente la phase de recherche finale. Parmi les cinq vérifications à tester dans le SDSP, la position qui donne le MCD permet d'obtenir le vecteur de mouvement du bloc donnant la meilleure correspondance.

L'algorithme DS est donc structuré comme suit :

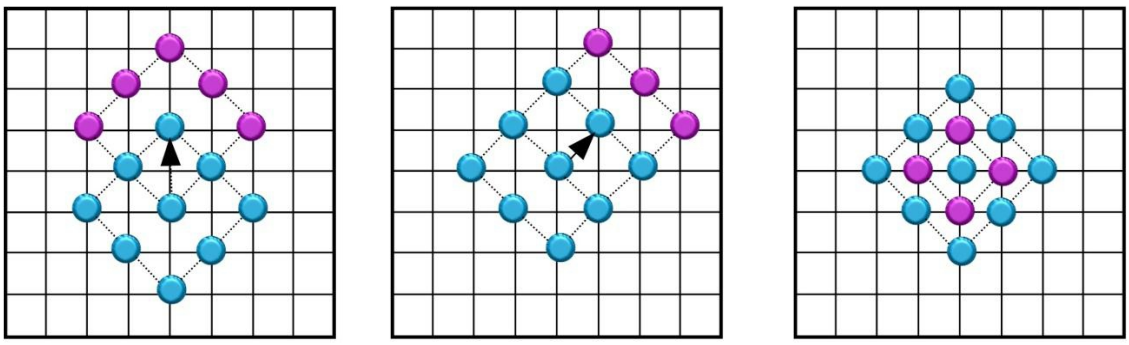
Etape 1 : Le LDSP initial est centré sur l'origine de la fenêtre de recherche, et les 9 positions du LDSP sont évaluées en termes de distorsion. Si le MCD obtenu est situé en position centrale, alors l'algorithme passe à l'étape 3, sinon il passe à l'étape 2.

Etape 2 : Le MCD obtenu dans l'étape de recherche précédente sert de centre à la nouvelle grille de type LDSP. Si le nouveau MCD obtenu est située à la position centrale, alors l'algorithme passe à l'étape 3, sinon, cette étape est répétée.

Etape 3 : Le motif de recherche passe de LDSP à SDSP. Le point trouvé dans cette étape représente la position correspondant au macro-bloc de référence la plus probable dans la zone recherche.

Les Figures 2.7 (a,b) illustrent les deux déplacements possibles de la grille LDSP de recherche. La Figure 2.7 (c) représente l'utilisation de la grille de type SDSP, consécutive à l'obtention du MDB en position centrale lors de l'itération précédente. Le chevauchement des

positions est représenté pour ces trois configurations. En violet, apparaissent les positions n'ayant pas été évaluées dans l'itération précédente.



a) b) c)

Figure 2. 7 : a) Déplacement consécutif à l'obtention du MCD en coin ;b) Déplacement consécutif à l'obtention du MCD sur un bord ; c) Passage du LDSP à SDSP consécutif à l'obtention du MCD en position central

La figure 2.8 illustre un exemple de chemin de recherche qui conduit au vecteur de mouvement (-4,-3) avec un nombre d'étapes de recherche égal à cinq : quatre étapes pour la phase à base de LDSP et une étape finale avec l'utilisation du SDSP.

Au total la recherche a été réalisée pour 24 points, soit respectivement neuf, cinq, trois, trois et quatre points de recherche au cours des 5 phases d'estimation.

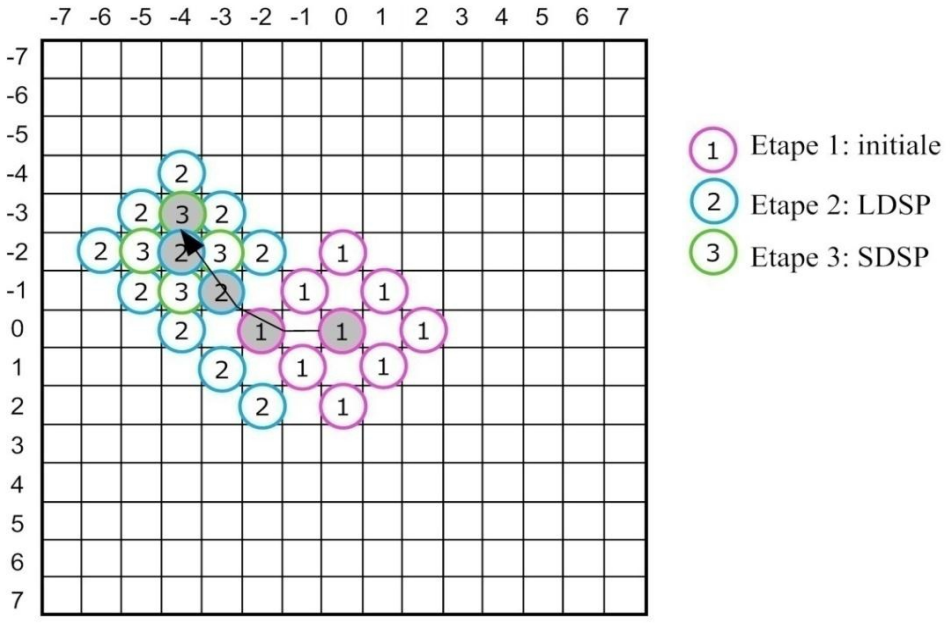


Figure 2. 8 : Exemple d'application de l'algorithme Diamond Search

2.2.4. Méthodes prédictives

D'une manière générale, le mouvement dans une scène vidéo implique plusieurs blocs et s'étale sur quelques trames. Il existe une corrélation spatiale et temporelle des champs de vecteurs entre ces blocs. En se basant sur cette observation, la fenêtre de recherche peut être restreinte à une zone plus petite, centrée autour des vecteurs prédits (figure 2.9).

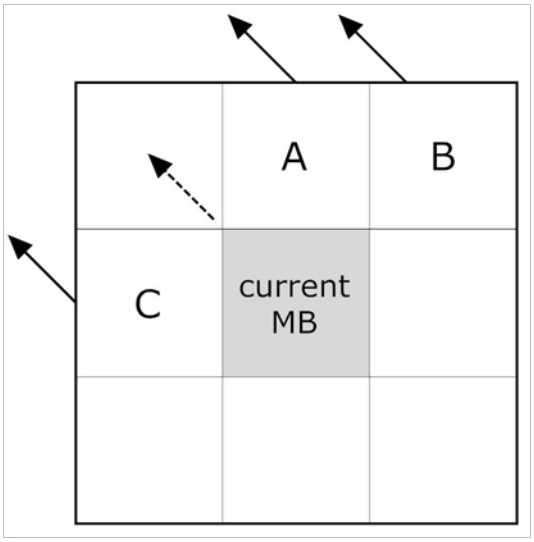


Figure 2. 9 : Prédiction de vecteur de mouvement

L'algorithme PMVFAST [TAL01a] (Predictive Motion Vector Field Adaptive Search Technique) illustre parfaitement ce principe. Celui-ci est basé sur l'utilisation des vecteurs estimés dans les blocs voisins. Il classe les mouvements en trois catégories: petits, moyens ou grands. A partir de cette classification, l'algorithme choisit une grille de recherche ayant une forme de grand ou de petit diamant. En outre, il crée un ensemble de points initiaux de prédiction, à partir desquels le meilleur point est sélectionné pour entamer la recherche. Le principal avantage est une convergence plus rapide qu'une recherche en diamant conventionnelle grâce à la phase de prédiction.

Plusieurs algorithmes ont été proposés et développés dans le contexte des stratégies de recherches prédictives [TAL01b][ZLC+04][XCY03]. L'estimateur de mouvement prédictif le plus populaire est l'EPZS (Enhanced Predictive Zonal Search) [Tou02]. Cet algorithme a été proposé dans le but d'améliorer son prédécesseur PMVFAST en introduisant un ensemble supplémentaire de prédicteurs plus efficaces (vecteur d'accélération, vecteur voisin). Une bonne sélection de prédicteurs semble être une caractéristique très importante pour l'amélioration de la vitesse par rapport aux méthodes précédemment mentionnées.

2.3.Etude analytique

Comme nous l'avons déjà mentionné, l'estimation de mouvement est considérée comme l'étape la plus gourmande en termes de temps de calcul dans un codage vidéo de type H.264/AVC. En outre, toute modification de cette étape influe directement sur l'ensemble des performances des codecs vidéo. Par conséquent, l'ajustement de la phase l'estimation de mouvement peut éventuellement permettre de s'adapter aux besoins des contraintes applicatifs (résolution d'image, frame-rate, débit binaire, PSNR).

Actuellement, différentes combinaisons peuvent être configurées dans le but d'optimiser les performances de codage. Selon notre étude, que nous détaillerons par la suite, les principales caractéristiques pouvant être ajustées au niveau de l'estimation du mouvement (figure 2.10) dans les normes actuelles telle que H. 264 sont les suivantes:

- L'algorithme de recherche utilisé,
- La taille des blocs (variable ou fixe),
- La précision d'estimation (pixélique ou subpixélique).

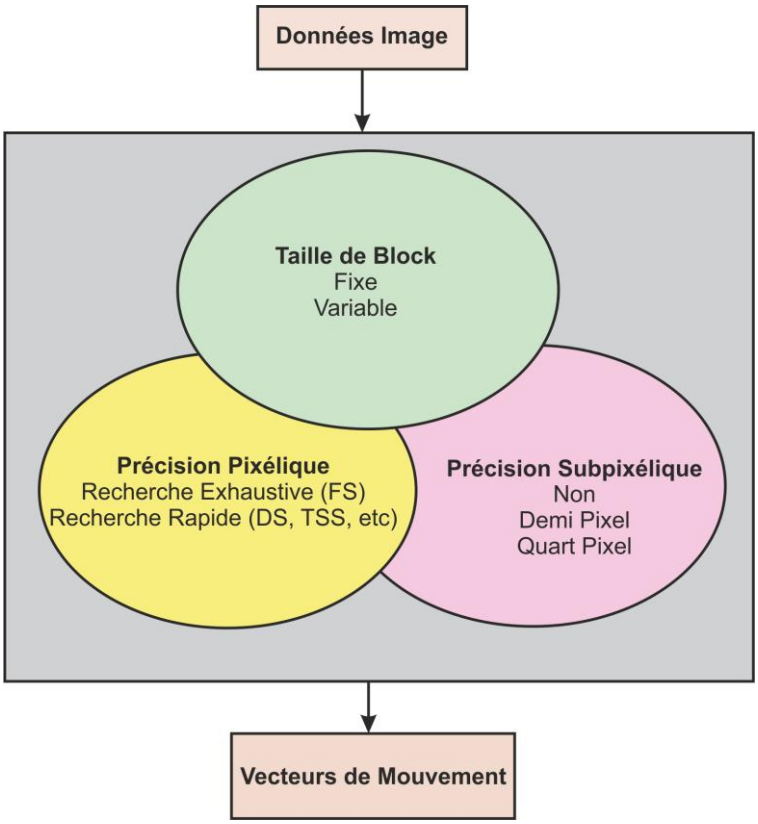


Figure 2. 10 : Paramètres de configuration

La grande diversité des standards de codage que nous avons apprécié au cours du premier chapitre, mais aussi celles des terminaux, des réseaux de communication, développent un besoin de très grande flexibilité des encodeurs vidéo. Parallèlement à ce besoin, les utilisateurs demandent toujours une qualité supérieure au niveau du rendu visuel en particulier pour les applications multimédia et mobiles. L'accélération matérielle représente une solution efficace pour répondre aux nouveaux défis de codage à haute performance, bien entendu à condition qu'elle présente une flexibilité suffisante afin d'ajuster les performances d'encodage (éventuellement dynamiquement d'ailleurs !) aux besoins de l'application.

C'est dans ce contexte que nous allons nous placer pour étudier l'impact des différentes techniques d'estimation de mouvement à la taille de bloc variable mais aussi de l'utilisation du raffinement subpixélique afin de proposer un accélérateur matériel d'un estimateur de mouvement configurable (et donc flexible) entièrement compatible avec la norme H.264/AVC.

2.3.1. Impact de la stratégie de recherche

Nous avons vu précédemment qu'il était possible d'accélérer considérablement l'estimation de mouvement grâce à des stratégies de recherche, non exhaustives, de la meilleure correspondance de bloc. Pourtant, une question se pose : est-il possible de conserver la même précision de codage en utilisant ces stratégies de recherche ? Bien entendu, comme c'est le cas généralement dans le domaine de la compression, le résultat dépend des vidéos à encoder. Pour autant, un certain nombre d'études dans la littérature permettent d'évaluer l'impact sur l'encodage de l'utilisation de ce type de stratégies par rapport à la recherche exhaustive. On peut notamment citer les travaux décrivant la recherche en trois étapes (TSS) [KIH+81], la nouvelle recherche en trois étapes (NTSS) [LZL94], la recherche en quatre étapes (4SS) [PM96], (HEXBS) [ZLC02], la recherche suivant une grille de diamant (DS) [ZM00], la recherche en diamant croisés (CDS) [CP02] , la recherche basée sur les blocs de descente de gradient (BBGDS) [LF96].

En se référant à ces travaux, il est possible de noter que ces algorithmes offrent de meilleurs résultats surtout dans les séquences vidéo à faible résolution. Toutefois, la réduction du nombre de « matching », peut générer une dégradation de la qualité d'image. Cet aspect est particulièrement important, c'est pourquoi nous avons réalisé une étude analytique générale

suivant les différents types ou natures des séquences vidéo, dans le but d'adapter le compromis entre temps d'exécution, qualité d'image et débit binaire.

L'amélioration de la qualité d'image et du débit binaire dépend notamment de la précision des positions des vecteurs de mouvements obtenus au cours de la recherche. Plus les vecteurs de mouvement convergent vers les meilleures positions, plus l'énergie résiduelle diminue, plus le débit binaire se réduit, offrant une meilleure compensation de mouvement. D'un autre côté, la réduction du temps d'estimation est obtenue en choisissant un des algorithmes de recherche rapides, qui consistent à réduire le nombre des tentatives de mise en correspondance, comparé à la recherche exhaustive.

Dans ce contexte, Lee et al. [LR06] ont présenté une étude générale permettant d'évaluer les différents algorithmes en fonction de différents types de mouvement (faibles, moyens, forts) présents dans les séquences vidéo. Cette étude est représentée Table 2.1.

				FS	BBGDS	DS	CDS	MVFAST	PMVFAST	HEXBS
sequences faibles et moyens mouvements	Carphone	CIF	PSNR	34.07	33.79	33.71	33.65	33.75	33.74	33.40
			NPE	869.33	12.29	15.32	13.17	5.44	5.57	13.48
	Foreman	CIF	PSNR	33.41	33.17	33.07	32.98	33.03	33.04	32.58
			NPE	869.33	14.08	16.34	14.33	6.13	5.60	14.10
		QCIF	PSNR	32.55	32.40	32.31	32.19	32.17	32.21	32.05
			NPE	782.21	10.46	13.31	10.56	5.05	4.50	11.45
	Mobil	SIF	PSNR	22.59	22.57	22.55	22.57	22.56	22.56	22.50
			NPE	859.45	10.71	12.57	10.13	4.98	4.66	11.96
sequences forts mouvements	Tennis-Table	SIF	PSNR	29.61	28.28	28.73	28.66	28.65	28.83	28.52
			NPE	859.45	12.58	15.47	13.75	5.73	5.50	13.28
		QCIF	PSNR	27.76	26.64	26.91	26.82	26.70	27.01	26.69
			NPE	782.21	10.88	13.89	11.82	6.71	4.97	12.56
	Football	SIF	PSNR	22.98	21.99	22.17	22.07	22.32	22.42	22.12
			NPE	859.45	14.64	16.75	15.00	8.21	7.34	14.21

Table 2. 1 : Impact des stratégies de recherche pour différents types de séquences vidéo [LR06]

L'utilisation d'un des algorithmes rapides avec les séquences vidéo contenant de faibles et de moyens mouvements (Carphone, Foreman et Mobile), permet de démontrer une très forte accélération. Dans la séquence Foreman (QCIF), le nombre des points évalués pour un macro-bloc (NPE) atteint 782 en utilisant la recherche exhaustive, cependant, lors de l'utilisation de l'un des algorithmes rapides (DS, BBGDS, CDS, PMVFAST, etc), le NPE ne dépasse pas 14 points et cela avec uniquement une légère dégradation au niveau qualité de l'ordre de 0.24 dB dans le cas du Diamond Search. Cependant, dans les cas des séquences

notées rapides contenant une grande quantité de mouvements (Tennis Table, Football), l'application de la majorité de ces algorithmes rapides cause une diminution notable du PSNR variant entre 0.56 dB (PMVFAST) à 1 dB (BBGDS) qui correspond à une dégradation visuelle importante de la qualité d'image. Malgré tout, les deux algorithmes DS et BBGDS fournissent une qualité plus élevée que celles des autres algorithmes rapides mais légèrement inférieure à celle obtenue avec le FS, tout en conservant le même ordre de grandeur au niveau de l'accélération en temps de traitement. Nous proposons Table 2.2, une représentation synthétique de cette même étude afin de mettre en évidence les grandes tendances de l'impact du type de mouvement ainsi que de la classe d'algorithme sélectionnée sur, d'une part la qualité d'image, et d'autre part le nombre de point testés. Pour cela nous proposons d'analyser les résultats en les regroupant en deux classes distinctes, à savoir ceux obtenus à partir de l'algorithme de recherche exhaustive (FS) et les résultats moyens obtenus pour l'ensemble des méthodes rapides. La comparaison est réalisée à partir des résultats en mode FS et pour les mouvements faibles et moyens. C'est en effet dans ce cas que la meilleure qualité d'image est obtenue. Cette configuration est normalisée à la valeur 100. Dans le cas des mouvements lents, il est clair que l'économie due aux algorithmes rapides est très significative en termes de calcul et cela pour une perte de qualité minime. On peut noter que la perte de qualité induite par les algorithmes rapides est plus importante pour les mouvements importants. Le concepteur aura a donc un choix à faire en fonction de l'application visée, choix qui sera dicté par le meilleur compromis entre qualité et temps de traitement. Ceci rendra judicieux la définition d'un modèle d'architecture commun aux deux familles d'algorithmes.

		Algorithme lent (FS)	Algorithmes rapides (DS, etc.)
Mouvement Faible et moyen	Ratio PSNR	100	99
	Ratio NPE	100	1.15
Mouvement importants	Ratio PSNR	87	84
	Ratio NPE	98	1.28

Table 2. 2 : Impact des stratégies avec une représentation de ratio normalisé

D'autre part au cours d'une étude du même type, Ismail et al. [IMS+09] ont proposé un algorithme rapide proche de la méthode DS : l'algorithme de recherche en diamant modifié (MDS). Les auteurs ont comparé leur méthode à différents algorithmes. L'étude montre que la

méthode proposée ainsi que les autres algorithmes rapides tels que DS, 4SS et N3SS offre une accélération remarquable du temps de traitement par rapport à la méthode de recherche exhaustive. Dans le cas d'une séquence rapide (football), cette accélération est de l'ordre de 99% (MDS), 94% (DS), 73% (4SS) et 65% (N3SS) comparativement aux méthodes citées précédemment. Une dégradation négligeable à la fois en PSNR et en débit binaire est observée.

Deux éléments importants sont mis en évidence par notre analyse de l'état de l'art. D'une part, la possibilité de modifier la stratégie de recherche pourrait permettre de s'adapter dynamiquement aux caractéristiques (telles que le type de mouvement par exemple) de la vidéo à encoder. Il faut noter que l'utilisation de la méthode de recherche exhaustive peut être considérée dans certains cas afin d'obtenir la meilleure qualité d'image possible ou de maximiser le taux d'encodage. D'autre part, la famille DS peut être considérée comme un compromis intéressant entre performance de codage et temps de calcul. De plus sa régularité, représente un atout intéressant pour une implantation matérielle.

En conclusion, une architecture matérielle flexible pouvant supporter un grand nombre de stratégies de recherche (itératives ou exhaustive) apparaît donc bien comme une solution pertinente vis-à-vis de l'hétérogénéité des performances d'encodage à fournir dans les systèmes de compressions actuels.

2.3.2. Impact de raffinement subpixélique et de la taille variable de blocs

Nous avons vu que H.264/AVC introduit deux nouvelles caractéristiques : la taille variable des blocs « Variable Blocs Size Motion Estimation » (VBSME) et le raffinement subpixélique des vecteurs de mouvements « Fractional Motion Estimation » (FME). Le VBSME est réalisé en deux phases: l'estimation de mouvement pixélique (EMP) et l'estimation de mouvement subpixélique (EMS). Cette dernière améliore la précision de prédiction et par conséquent aussi la qualité d'image par rapport à la technique classique à taille fixe. Cependant, le VBSME [Wie03**]** augmente de manière importante la quantité de calcul nécessaire.

Comme on l'a mentionné dans le chapitre précédent, chaque macro-bloc peut se décomposer en différentes partitions de tailles (4x4, 4x8, 8x4, 8x8, 8x16, 16x8, 16x16). Selon les mouvements de ces sous blocs on obtiendra jusqu'à 41 vecteurs de mouvements. Si les directions de mouvement sont homogènes, un fusionnement des partitions est alors opéré.

Par ailleurs, les mouvements des blocs ne correspondent pas toujours à une position entière exacte. Une estimation fractionnaire peut donc être réalisée pour améliorer l'estimation. Dans ce cas, une interpolation subpixel du bloc référence est alors réalisée.

Selon Yang et al. [YGI06] et Chen et al. [CHC04][CCC+08], le raffinement des vecteurs de mouvements à la précision demi et quart de pixel apporte un gain d'environ +4dB du PSNR. La dégradation de la qualité des images peut aller jusqu'à 60%, lorsque le raffinement subpixélique n'est pas activé. Les performances des estimateurs de mouvement au quart de pixel sont visiblement meilleures qu'au pixel entier. Les résidus sont réduits, donnant une image de meilleure qualité à un débit plus faible. En revanche une charge de calcul supplémentaire conséquente de l'ordre de 40% en temps de prédiction est introduite.
L'estimation de type VBSME apporte un gain inférieur à celui obtenu avec la précision subpixélique. En général, une estimation de type VBSME est donc associée à l'estimation subpixélique.

Par conséquent, un estimateur de mouvement supportant les estimations VBSME et sub-pixels permet d'améliorer les performances de codage. Une architecture configurable permettant d'ajuster les stratégies de recherche et les types de raffinements représente donc une solution efficace pour répondre à une certaine combinaison ou configuration demandées par les utilisateurs.
Avant de proposer notre propre architecture configurable, nous allons maintenant passer en revue les architectures principales existantes, permettant l'estimation de mouvement, d'abord au niveau pixélique, puis subpixélique.

2.4. Architectures matérielles d'estimateur de mouvement pixélique

Plusieurs architectures ont été proposées et développées. Les plus connues s'intéressent aux algorithmes de recherche exhaustive à taille de blocs variables ou fixes, soit respectivement VBSME et FBSME. Comme nous l'avons vu, dans le cas d'une recherche exhaustive le nombre de positions de recherche est très important. Ainsi la plupart de ces architectures sont de type systolique afin de diminuer le temps de calcul. Une alternative communément rencontrée est l'implantation d'architectures supportant des algorithmes de recherche rapide.

2.4.1. Architecture systoliques de type FS

Il existe différentes architectures VLSI d'estimateur de mouvement au niveau pixel, supportant la stratégie FS. Les architectures matérielles traditionnelles d'estimateurs sont conçues à taille de bloc fixe FBSME et peuvent être classées en deux catégories. Une architecture Inter-Candidat, où chaque élément de traitement ou « Processing Element » (PE) est responsable du calcul de la valeur SAD d'un candidat de recherche spécifique, et l'autre est une architecture Intra-Candidat, où chaque PE est responsable du calcul de la distorsion d'un pixel courant spécifique dans le macro-bloc courant, pour tous les candidats de recherche.

Les architectures sont couramment proposées afin d'accélérer le temps de calcul du MCD nécessaire à la recherche exhaustive de toutes les positions du motif dans la zone de recherche. Ces architectures sont composées d'un ensemble de processeurs élémentaires interconnectés sous forme d'un vecteur à une dimension (1D) ou à deux dimensions (2D). Dans une architecture 1D, chaque PE est connecté à ses deux voisins horizontalement ou verticalement et dans le cas 2D avec ses quatre ou huit voisins. Ce type d'architectures est bien entendu très coûteux en ressources matérielles. Dans cette partie, nous présentons trois principales architectures matérielles d'un estimateur de mouvement FBSME [TMK00][CCH+06] proposées par Yang et al [YSW89], Yeo et Hu [YH95] et Komarek et Perish [KP89]. Ces trois architectures sont particulièrement intéressantes, et constituent la brique de base de nombreuses architectures matérielles développées et proposées par la suite.

Dans la discussion qui suit, nous supposons que la taille de bloc est NxN et que la fenêtre de recherche est limitée à un déplacement [-Ph, Ph] et [-Pv, Pv] dans les deux directions horizontale et verticale.

2.4.1.1. Architecture systolique 1D de Yang et al

Yang et al [YSW89] ont mis en œuvre la première architecture VLSI d'un estimateur de mouvement. C'est une architecture 1D, comme le montre la figure 2.11. Chaque élément de traitement (PE) est chargé de calculer le critère de distorsion entre les pixels candidats et références. Le nombre de PE est égal au nombre de blocs de références suivant la direction horizontale de la fenêtre de recherche. Les pixels des blocs courants sont propagés à travers des registres à décalage tandis que les pixels du bloc de référence sont diffusés à tous les PE. Cette architecture permet une entrée séquentielle des données, mais elle effectue un traitement parallèle avec un taux d'utilisation du matériel de l'ordre de 100%. En outre, cette conception

permet de réduire l'accès à la mémoire par la diffusion et la propagation des pixels du bloc de référence. Toutefois, cette architecture est considérée comme lente surtout pour une large zone de recherche, ce qui a poussé les chercheurs à augmenter les nombre de PE dans les deux directions (2D). Ce type d'architecture est décrit dans le paragraphe suivant.

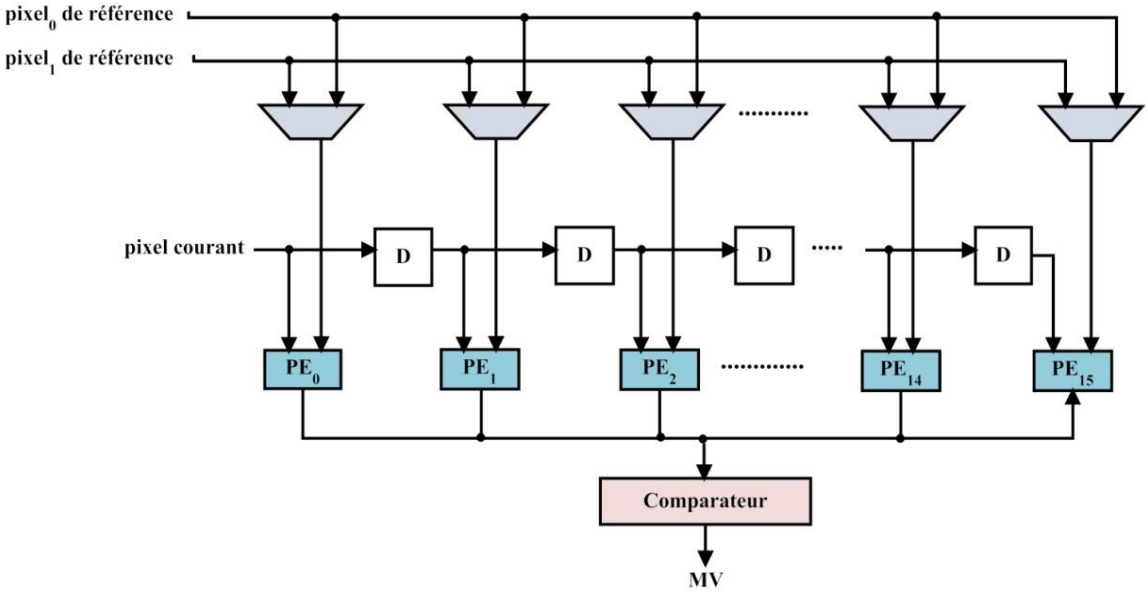


Figure 2. 11 : Architecture de Yang 1D

2.4.1.2. Architecture systolique 2D de Yeo et Hu

La Figure 2.12 présente une architecture matérielle Inter-Candidat 2-D, proposé par Yeo et Hu [YH95] Et composée par (NxN) PE. Elle est similaire à l'architecture précédente de Yang. Les pixels de référence sont diffusés dans les PE, et les pixels courants sont propagés à travers les registres de propagation noté D dans la Figure 2.12. Les valeurs du SAD sont respectivement stockées et accumulées dans les PE. En raison de la diffusion des pixels de référence dans les deux directions, le nombre de PE doit correspondre à la taille des macro-blocs. La caractéristique de cette architecture est la diffusion des données dans deux directions en même temps, ce qui peut augmenter la réutilisation de données. Cette architecture est considérée comme étant rapide par rapport à celle 1D proposée par Yang [YSW89] mais reste toujours incapable de gérer les contraintes temps réel imposées par l'augmentation constante des besoins de la technologie multimédia, ce qui explique l'augmentation du nombre de PE dans d'autre travaux [VS89][RS02].

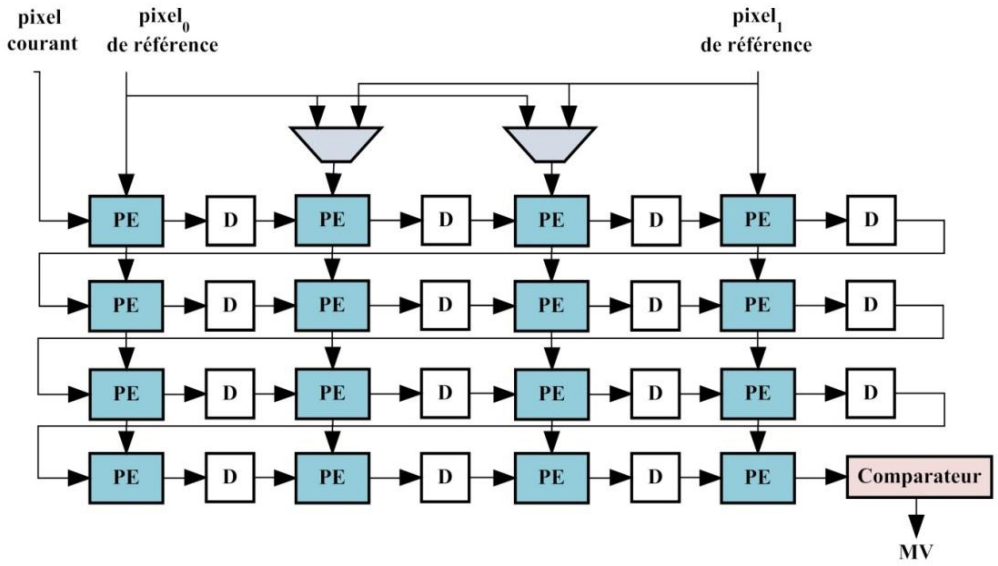


Figure 2. 12 : Architecture 2D de Yeo et Hu

2.4.1.3. Architecture systolique 2D de Komarek et Pirch

Komarek et Pirsch [KP89] ont proposé une architecture systolique 2-D intra-candidat, comme indiqué dans la figure 2.13.

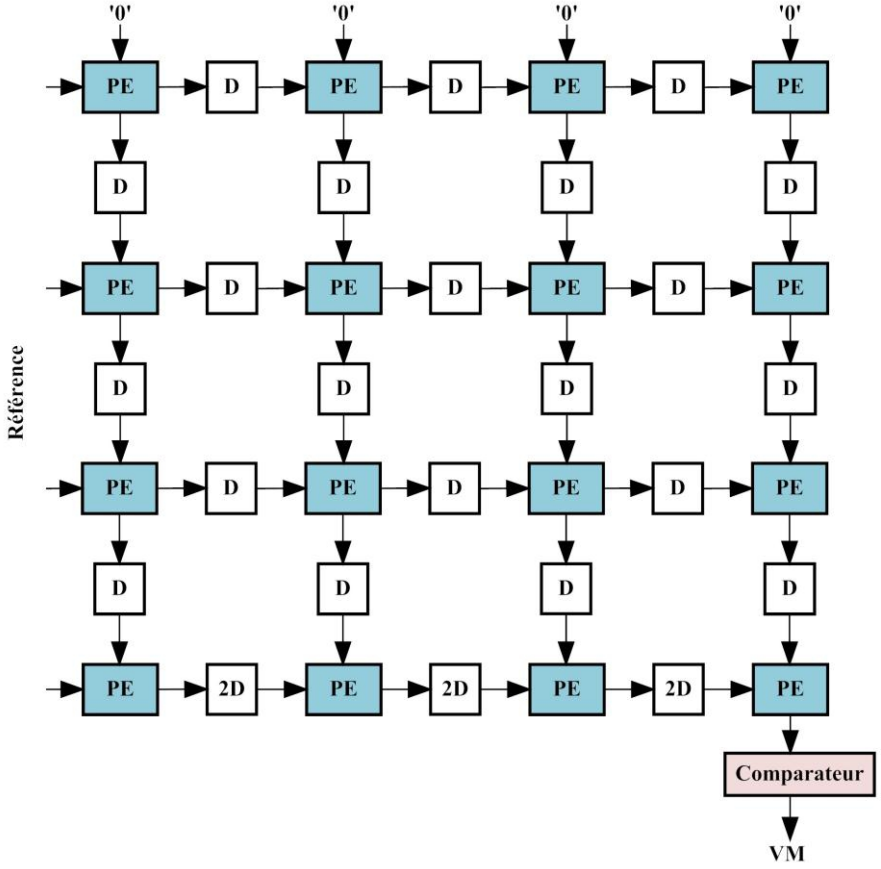


Figure 2. 13 : Architecture 2D array de Komarek and Perish

Les pixels courants sont stockés dans les PE correspondants. Les pixels de référence sont propagés horizontalement à travers les PE. Ensuite, Les valeurs SAD de chaque colonne sont propagées et accumulées dans la direction verticale dans un premier temps. Après propagation verticale, les valeurs accumulées SAD se propagent dans une direction horizontale.

Les architectures systoliques permettent de supporter les stratégies et les algorithmes réguliers. C'est à dire que l'accès à la mémoire ne doit pas être aléatoire pour pouvoir profiter de la technique de décalage. Pour cette raison, les travaux que nous venons de présenter sont restreints à l'algorithme de recherche exhaustif. Pour les algorithmes rapides, d'autres techniques doivent être mises en œuvre.

2.4.2. Architecture pour taille de bloc variable

Plusieurs architectures [YM04][LKK05][GHJ05][SLI+06][FAS09] concernant l'estimation de mouvement à taille de bloc variable ont été proposées et implémentées. La majorité de ces implémentations sont associées à l'algorithme de recherche exhaustive. La solution la plus souvent adoptée est la réutilisation de SAD. Cette technique consiste à calculer les SAD des plus petits sous-blocs de taille 4x4, et en calculant simultanément tous les sous-blocs inclus dans un bloc de la plus grande taille, c'est à dire 16x16. L'insertion d'un arbre d'addition permet d'obtenir les SAD des plus grands blocs en combinant les résultats de niveau inférieur. Cette technique de réutilisation SAD peut considérablement réduire la complexité du calcul.

La solution proposée par Yap est la plus populaire. C'est une architecture systolique 1D utilisant 16 PE, basée sur le même flot des données que celui présenté par Yang et al. Dans le but d'accélérer le calcul, plusieurs travaux ont été proposés, basés sur une architecture systolique à deux dimensions (2D), tout en augmentant le nombre des PE utilisés [LKK05]. Bien entendu, l'utilisation d'une architecture systolique a pour conséquence une augmentation importante des ressources matérielles nécessaires par rapport à une architecture classique, mais permet aussi une accélération notable des traitements.

Ces architectures ont été proposées avec la stratégie de recherche exhaustive avec une précision pixélique. Ces architectures ne peuvent pas supporter différentes stratégies de recherche. Il n'existe pas d'architecture uniforme répondant aux contraintes des différents algorithmes. Les architectures actuelles ont été développées pour un algorithme spécifique.

Nous focalisons maintenant notre présentation de l'état de l'art sur les architectures matérielles supportant les méthodes possédant un nombre réduit de positions: les algorithmes de type « Fast Search ».

2.4.3. Architecture des Algorithmes rapides : cas du Diamond Search

Comme nous l'avons décrit précédemment, pour réduire le nombre de mises en correspondance à réaliser, on utilise une grille de recherche dans laquelle généralement au maximum 9 positions seront testées par phase d'estimation. Ainsi la plupart des implantations proposent l'utilisation d'architectures parallèles de 9 PE. Cependant, ces algorithmes nécessitent une gestion des données optimisée dans le but de réduire le temps de traitement et la consommation. Dans ce contexte, plusieurs travaux se sont focalisés sur la gestion de la mémoire. Dans la suite, nous détaillerons une sélection d'architectures matérielles supportant une stratégie de type « Diamond Search ».

Il existe différents algorithmes définis à partir du DS, on peut notamment citer: Centre-Based DS [DY11], Multi-Point Diamond Search (MPDS) [SNP+11], Hardware-Oriented Modified Diamond Search [NO11], Line Diamond Parallel Search (LDPS) [KLA+11].
L'algorithme de recherche MPDS [SNP+11] utilise le même principe que l'algorithme DS. Cependant, la recherche n'est pas faite seulement au centre de la zone de recherche. L'algorithme MPDS trouve la meilleure adéquation dans cinq positions différentes d'une zone de recherche. Chaque position, sauf celle du centre, est définie à l'intérieur d'un secteur (A, B, C et D). Cependant, cet algorithme nécessite cinq cœurs de traitement de type DS indépendants qui seront déclenchés dans la zone de recherche. Le MPDS n'est pas restreint à un seul point de départ, précisément pour éviter le même minimum local.

L'algorithme MPDS travaille avec des blocs de 16x16. Une architecture a été conçue pour exécuter une recherche de type DS et a été utilisée comme support pour permettre en œuvre l'algorithme MPDS de ME. L'architecture de base tout d'abord est présentée dans le cas d'une utilisation avec un algorithme de type DS et puis son adaptation, supportant une méthode de type MPDS, est ensuite proposée.

2.4.3.1. Architecture de base de DS

La figure 2.14 présente le schéma de l'architecture de base de l'algorithme DS. Cette architecture comporte deux bancs mémoires permettant respectivement le stockage de la zone

de recherche et du macro-bloc courant. La taille adoptée pour la fenêtre de recherche est de 34x34 pixels.

Chaque cœur en diamant requiert 34 cycles pour assurer le remplissage de la mémoire interne (fenêtre de recherche) correspondante. Les mémoires de MEM1 à MEM9 sont dédiées au stockage des pixels utilisés pendant la phase LDSP. Chaque mémoire doit contenir les pixels d'un macro-bloc de taille 16x16 correspondant aux 9 points de recherches (LDSP). De la même manière, les mémoires MEMA, MEMB, MEMC et MEMD sont utilisées pour stocker les données nécessaires des quatre macro-blocs requis durant la phase finale SDSP.

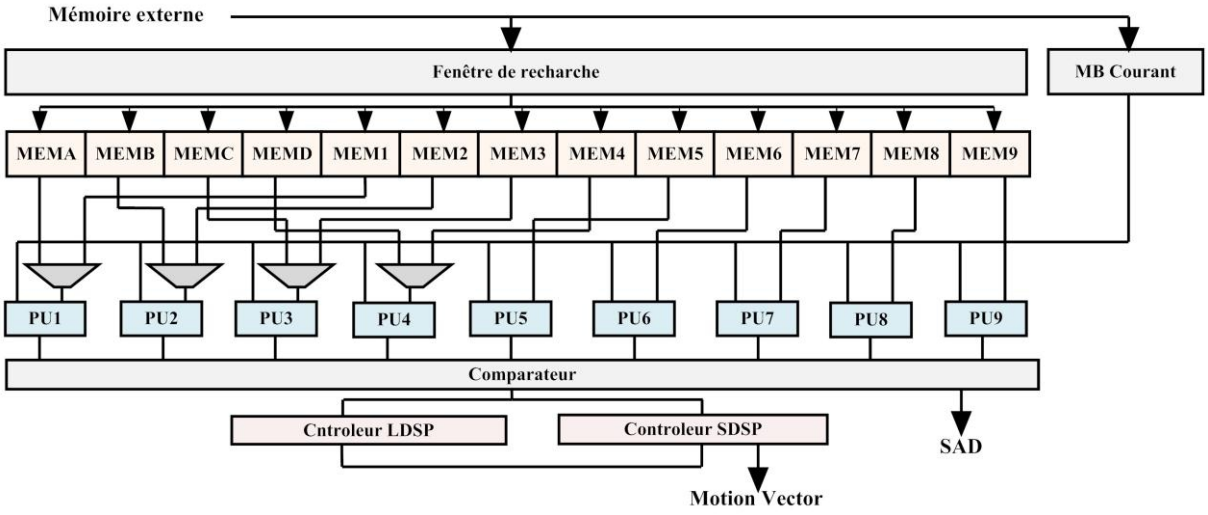


Figure 2. 14 : Schéma de l'architecture de base DS

L'architecture de base de DS est composée de neuf unités de traitement notées « Processing Unit » (PU). Chaque PU peut être en charge du calcul d'une valeur SAD pour un macro-bloc donné de taille 16x16. Toutes ces unités de traitement sont directement connectées à une des 9 mémoires consacrées pour le traitement LDSP, ce qui favorise un traitement parallèle pour tous les points du diamant. Ces 9 unités partagent la mémoire du macro-bloc courant tels que présenté sur la figure 2.14.

Dans le but d'optimiser la consommation du système, la phase SDSP n'active que les quatre premières unités de traitement pour assurer le calcul final, ce qui explique la nécessité d'un ensemble de multiplexeur permettant la sélection des données valides.

Chaque PU est un pipeline de cinq étages où sont réalisées les additions nécessaires au calcul de la valeur de SAD.

Le comparateur retourne l'adresse du macro-bloc ayant obtenu la valeur minimale du critère de distorsion. La sortie du comparateur est mise à jour à chaque étape. Si le meilleur bloc central à savoir PU5 obtient le meilleur résultat, la phase SDSP est déclenchée. De plus, le contrôleur de position SDSP génère le vecteur de mouvement final pour ce macro-bloc. Dans le cas où la meilleure correspondance n'est pas obtenue en position centrale, une nouvelle itération est déclenchée en phase LDSP. Dans le pire des cas, ce co-processeur génère un vecteur de mouvement en 169 cycles d'horloge.

Cette architecture, où différentes positions de recherche sont estimées en parallèle, n'est pas très économique d'un point de vue ressources matérielles mais permet d'optimiser le calcul d'une phase d'estimation. L'exploitation maximale du parallélisme de données est un concept pouvant être transposé au calcul des valeurs subpixéliques mais il faut aussi noter qu'elle n'est rendu possible qu'avec une optimisation des accès aux données, à savoir un accès en parallèle aux données de la zone de recherche.

2.4.3.2. Architecture de base de MPDS

La figure 2.15 représente le schéma général d'architecture MPDS. Chaque élément de traitement, noté CORE, est composé d'une architecture de base DS évoquée précédemment.

Tout d'abord l'unité centrale (Core C) commence le traitement par le remplissage des mémoires internes, suivi par l'évaluation des macro-blocs du diamant. Par la suite, avec une latence de 34 cycles les unités (Core 1, Core 2, Core 3, Core4) sont déclenchés l'un après l'autre en gardant cette latence de 34 cycles. Le comparateur est en charge de déterminer le meilleur SAD (minimum) parmi les meilleurs SAD des cinq unités CORE.

La version proposée de l'architecture supportant la méthode MPDS possède une latence mémoire de 170 cycles nécessaires au remplissage de toutes les mémoires stockant les données de références. De plus, 135 cycles sont nécessaires pour que les 4 cœurs terminent leurs traitements, et 15 cycles supplémentaires sont nécessaires pour terminer le SDSP. En outre, plus de trois cycles sont nécessaires pour déterminer le meilleur résultat entre les différentes unités. Une approche séquentielle nécessiterait 323 cycles pour la génération du vecteur de mouvement. Les auteurs proposent une structure en pipeline de ces tâches. Ainsi, seulement 170 cycles sont nécessaires pour générer un vecteur de mouvement. Cette architecture est rapide au niveau de traitement, cependant son coût important en ressources mémoires reste un inconvénient majeur pour une implantation matérielle (notamment pour une cible de type FPGA).

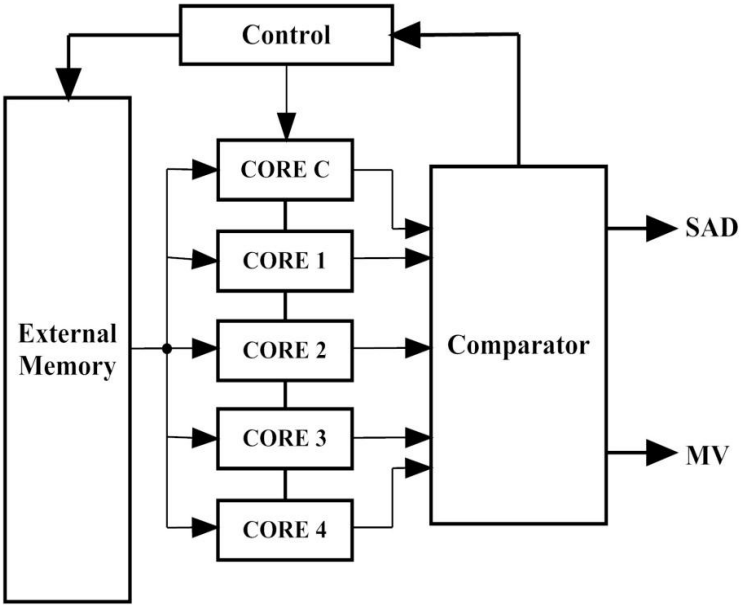


Figure 2. 15 : Schéma de l'architecture MPDS.

2.5. Architectures matérielles d'estimateur de mouvement subpixélique

L'estimation de mouvement subpixélique consomme plus de 40% du temps d'estimation. L'accélération matérielle de ce traitement basé sur l'exploitation des différents parallélismes potentiels de ces algorithmes de recherche nécessite d'importantes ressources matérielles. Ceci explique la grande variété d'architectures dédiées à ce type de traitement. Chacune d'entre elles cherche à obtenir le meilleur compromis possible entre temps de calcul et ressources matérielles utilisées. Dans la suite de cette présentation, nous décrivons les principales architectures extraites de la littérature, et à partir desquelles nous avons basé nos propres architectures.

2.5.1. Architecture de Chen

L'architecture de Chen est une des principales architectures d'estimateur de mouvement subpixélique [CHC04]. Elle est composée d'une unité d'interpolation, de neuf unités de traitement (PUi), de neuf unités d'accumulation de type SATD, une unité de contrôle de la phase SATD et une unité de comparaison. D'une manière générale, l'architecture globale d'un estimateur de mouvement subpixélique dépend de l'unité d'interpolation utilisée pour le raffinement demi-pixel. Chen et al ont proposé une unité d'interpolation de largeur 10 pixels, permettant l'interpolation d'un macro-bloc de taille 4x4. En effet, habituellement un filtre FIR

de taille 6 est utilisé pour réaliser ce calcul, et pour interpoler une ligne de N pixels, il faut accéder à (N+6) pixels, ce qui explique la largeur de 10 pixels.

Le raffinement des vecteurs de mouvement subpixélique est effectué pour tous les modes (4x4, 4x8, 8x4, 8x8, 8x16, 16x8, 16x16). Le bloc de taille 4x4 nommé « élément bloc » est le plus petit élément pour calculer la valeur SATD en H.264/AVC. L'unité d'interpolation présentée ne permet pas de traiter les sous-blocs de largeur supérieure à 4. Chen et al. proposent de les décomposer en sous-blocs de largeur 4 afin de les interpoler séquentiellement. De la même façon, les unités de traitements sont conçues pour calculer la SATD d'un bloc 4x4, afin de les réutiliser dans tous les types de blocs. Comme illustré figure 2.16, le sous-bloc de taille 4x8 est décomposé en deux blocs 4x4. La valeur SATD de deux blocs sont accumulés pour obtenir celle de sous-bloc 4x8. Le temps de traitement d'un élément bloc est de 10 cycles d'horloge. Le taux d'utilisation de ressources matérielles est de l'ordre de 100 % quelle que soit la taille des sous-blocs puisque conçue et optimisée pour ceux de taille 4x4. Toutefois le taux de calculs redondants est un peu élevé (18 % maximum pour des blocs de taille 16x16). Cette architecture permet de raffiner les 41 vecteurs de mouvement dans 1646 cycles d'horloge. C'est une architecture séquentielle qui minimise les ressources matérielles requises. Pour cette raison, cette architecture et notamment son unité d'interpolation sont largement réutilisées dans la littérature [RM10].

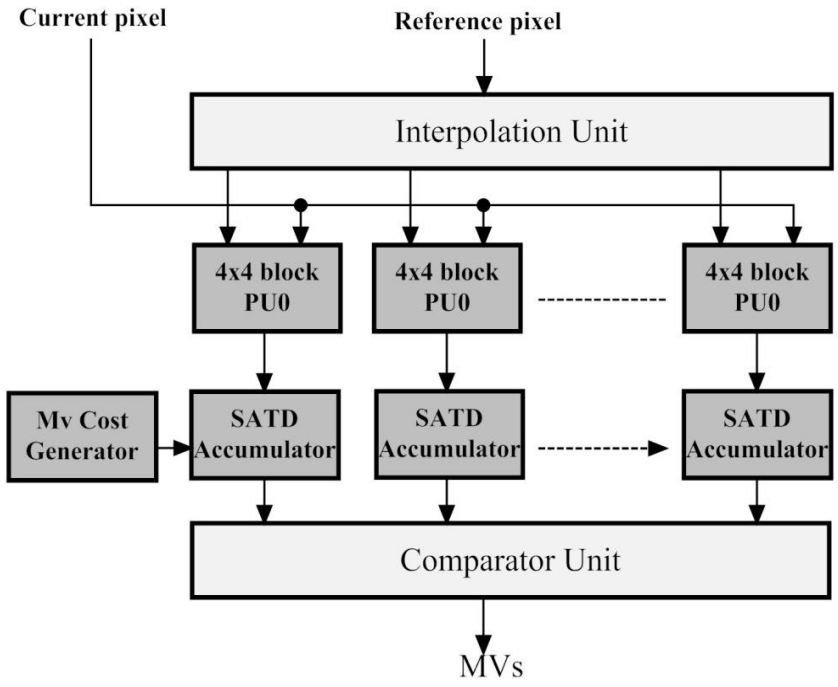


Figure 2. 16 : Architecture d'estimateur de mouvement subpixélique de Chenet al.

2.5.2. Architecture de Yang

Cette architecture a été proposée par Yang et al. dans le but d'accélérer le processus de raffinement subpixélique [YGI06]. L'idée principale est d'élargir l'unité d'interpolation (22 pixels en entrée) permettant d'interpoler, en un seul cycle, une ligne de 16 pixels à la fois. Cette architecture est décomposée en plusieurs unités comme représenté figure 2.17 : des unités d'interpolation demi et quart de pixel, des unités de traitement de SAD de taille 16x16, des registres, des multiplexeurs et des unités de contrôles afin commander le fractionnement en sous-blocs (4x4, 4x8, …, 16x16) et le niveau de raffinement (demi ou quart de pixel).

Comme mentionné précédemment, l'interpolation est une opération clef pour l'estimation subpixélique. En raison de sa contribution importante au temps de calcul global, toute accélération de la phase d'interpolation permet de réduire significativement ce temps. Ainsi Yang et al. ont choisi d'interpoler toute une ligne de 16 pixels pendant un cycle d'horloge. Pour le traitement d'un macro-bloc de taille 16x16, une zone de 22x22 pixels est nécessaire. Par conséquent 22 cycles sont requis pour interpoler ce macro-bloc.

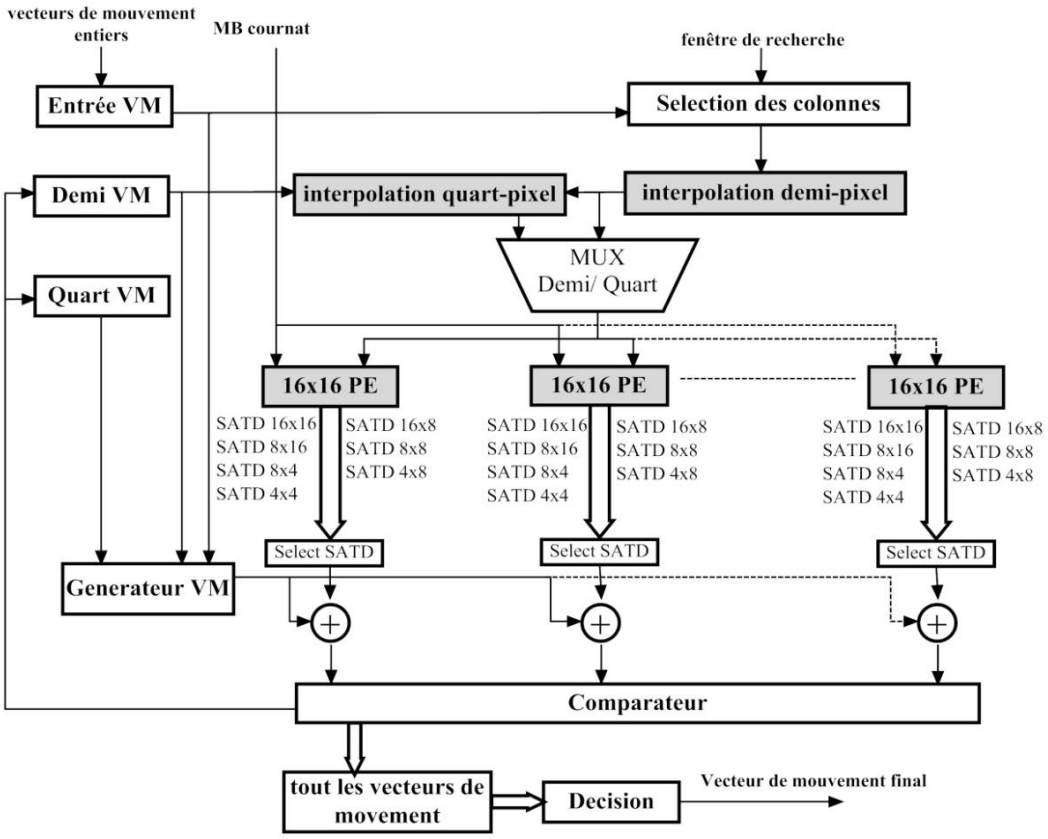


Figure 2. 17 : Architecture d'estimateur de mouvement subpixélique de Yang et al.

D'une manière générale, à partir d'une unité d'interpolation travaillant sur des lignes de M pixels, le nombre de cycles nécessaires pour interpoler un bloc de taille NxM est de (N+6).

Dans l'architecture proposée, cette unité d'interpolation (de 20 pixels en entrée) permet d'interpoler simultanément deux blocs de largeur 4, offrant une accélération en temps d'interpolation de l'ordre de 50%. Une duplication de la fenêtre de recherche est nécessaire pour traiter deux blocs en parallèle. Cette architecture permet une interpolation pour toutes les tailles sans produire de calculs redondants. Cependant, le taux d'utilisation de ressources matérielles est de l'ordre de 60%. L'utilisation de deux unités en parallèle ainsi que l'élargissement de l'unité d'interpolation réduit de près de 50 % le temps de traitement par rapport à l'architecture de Chen. Toutefois, les ressources matérielles nécessaires sont multipliées par trois.

2.5.3. Architecture de Ruiz

Afin de conserver une implantation matérielle à faible coût, Ruiz et al. [RM10] ont basé leur travail sur la même unité d'interpolation que celle présentée par Yang [YGI06]. La contribution majeure de Ruiz est relative au traitement du raffinement quart de pixel. Dans les architectures de Yang et de Chen, les raffinements demi et quart de pixel sont réalisés d'une manière séquentielle. Comme présenté dans la figure 2.18, Ruiz et al. ont proposé une architecture composée de 3 processeurs respectivement en charge du raffinement demi-pixel, du raffinement quart-pixel et de la phase décision qui se charge de fournir le vecteur de mouvement final pour chaque macro-bloc.

Ces trois processeurs traitent les données en pipeline. Ce type de structure nécessite l'ajout d'une unité supplémentaire assurant le contrôle du traitement global. Alors que l'architecture de Chen traite un macro-bloc dans 1664 cycles, le pipeline introduit par Ruiz réduit le temps jusqu'à 870 cycles.

Comme nous le décrirons dans la section 2.5, le raffinement quart de pixel permet d'améliorer significativement les performances de codage. Cette architecture présente un bon compromis entre le temps de traitement et ressources matérielles nécessaires afin d'atteindre une compression temps réel à haute résolution comme par exemple pour la télévision à haute définition (HDTV).

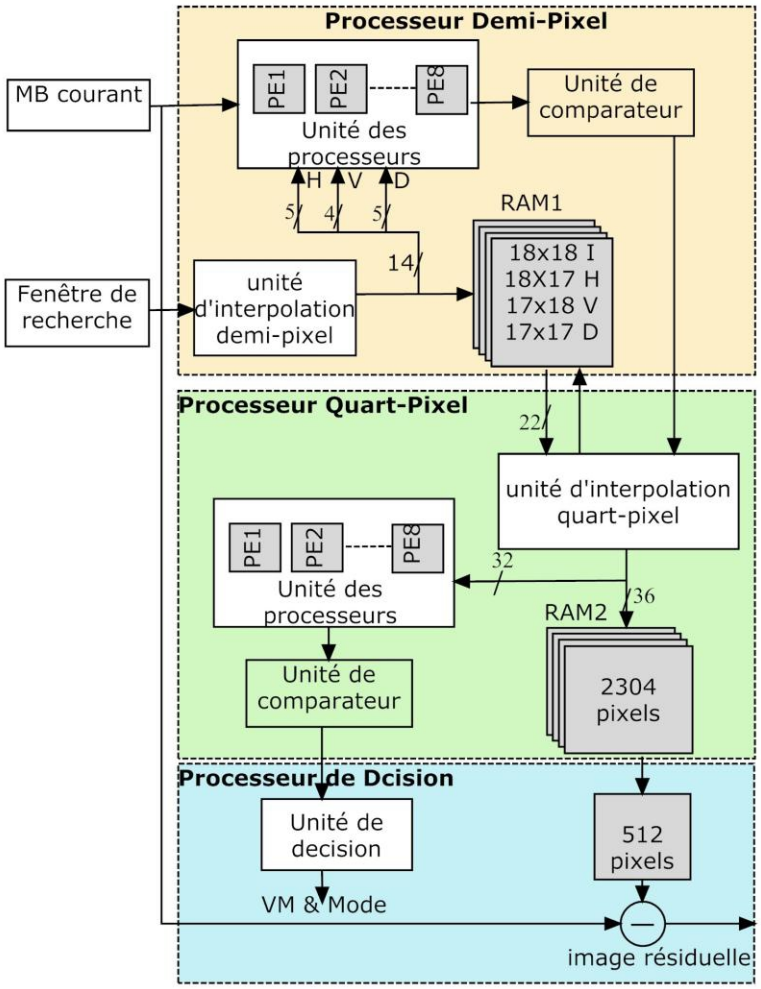


Figure 2. 18 : Architecture d'estimateur de mouvement subpixélique de Ruiz

2.5.4. Architecture de Thang

Thang Ta et al. ont proposé une nouvelle architecture basée sur une unité d'interpolation de largeur 14 pixels permettant d'interpoler une ligne de 8 pixels en un cycle [TC11]. Cette solution constitue donc une alternative possédant une complexité intermédiaire entre les deux architectures décrites précédemment. Comme montre la figure 2.19, cette architecture est composée d'une unité d'interpolation, d'une unité de contrôle, de six unités de processeur de taille 8x4, et d'une unité de comparaison.

L'unité d'interpolation utilisée permet d'interpoler tous les sous-blocs en 552 cycles. On peut noter que le taux d'utilisation des ressources matérielles peut atteindre atteint 100 % pour les blocs de largeur 8 et 16 pixels mais décroit à 57% pour les blocs de type 4x4. Les sous-blocs de largeur 4 (4x4 et 8x4) peuvent être traités simultanément, ce qui peut diminuer le temps de traitement.

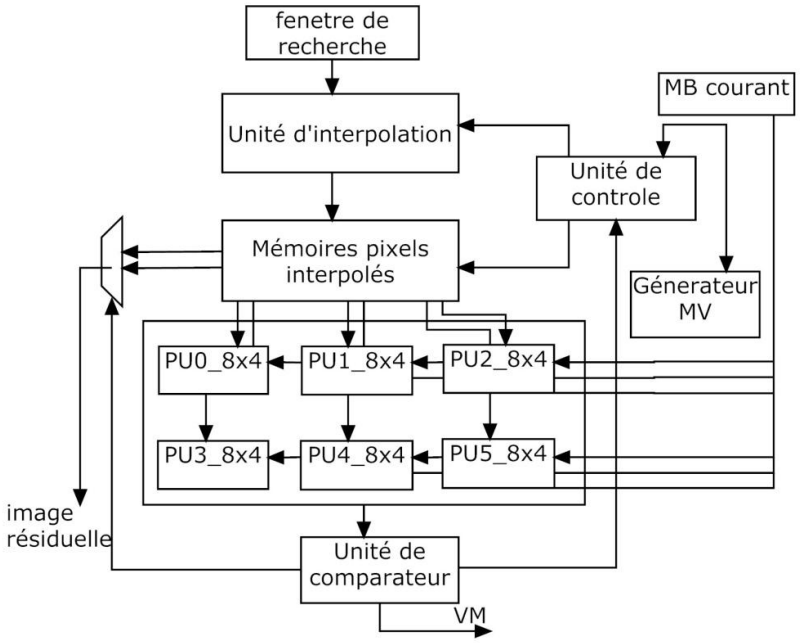


Figure 2. 19 : Architecture d'estimateur de mouvement subpixélique de Thang et al.

2.5.5. Architecture d'Urban

Urban et al. [URB07] ont proposé un coprocesseur subpixélique destiné aux systèmes à bande passante réduite (2 à 4 pixels par cycle) [UPD+06][UPN+07]. Cette architecture est configurable dans le sens où le parallélisme peut être augmenté (afin d'obtenir une plate-forme plus performante) et le filtre d'interpolation peut facilement être modifié. La taille du bloc à traiter est modifiable dynamiquement afin de conserver le maximum de flexibilité pour traiter les multiples tailles de sous-bloc et les unités de calcul sont dynamiquement configurables pour obtenir le maximum de performances. En comparaison avec les architectures précédentes, cette architecture s'avère moins rapide cela étant due, bien sûr, à son objectif d'utilisation pour des systèmes à bande passante réduite. Elle réduit cependant significativement les ressources matérielles.

Le coprocesseur est basé sur une structure en pipeline composé d'une unité d'interpolation, d'une matrice de processeurs élémentaires (PE), et d'un arbre de décision (Figure 2.20). Les PE calculent les distorsions pour chaque position candidate. La mesure de distorsion implémentée est la SAD car elle nécessite beaucoup moins de ressources qu'une SATD. La taille de bloc variable est supportée grâce à des paramètres de taille du bloc, modifiable dynamiquement.

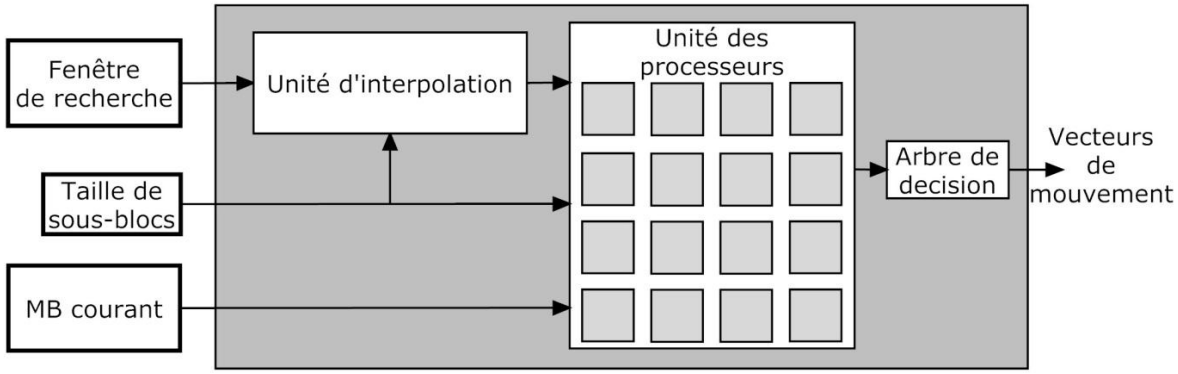


Figure 2. 20 : Architecture d'estimateur de mouvement subpixélique d'Urban

2.6. Architecture configurable d'estimateur de mouvement

Comme déjà décrit dans les sections précédentes, l'algorithme de recherche exhaustive est considéré comme la stratégie la plus régulière, étant donné que le nombre ainsi que les positions des points à tester sont fixés par la taille de la zone de recherche. Le séquencement du déplacement entre points est très simple puisque deux positions consécutives sont adjacentes ce qui simplifie les accès à la mémoire et facilite la réutilisation des données entre deux « matching ».

Les algorithmes rapides ne sont pas aussi réguliers. Le nombre des points à estimer, le nombre d'étapes ne sont pas toujours connus ; la grille de recherche peut évoluer au cours de l'estimation (DS, 4SS, etc). Parfois le nombre de points est connu mais les points de chaque étape de recherche sont répartis d'une manière irrégulière (TSS), ce qui entraîne des difficultés en termes d'accès mémoire.

Toutefois, en analysant de nombreux algorithmes rapides, on peut noter que l'ensemble de ces algorithmes peut se décomposer en plusieurs étapes répondant au même modèle. A chaque étape ou itération, un ensemble de positions de recherche est estimé. La position du minimum de distorsion dans la grille de recherche considérée permet soit de définir l'ensemble ou la liste de positions à estimer pour l'itération suivante soit de déterminer le vecteur de mouvement résultat. La génération des adresses (correspondant aux positions où l'on doit réaliser l'estimation de la distorsion) reste propre à chaque stratégie sélectionnée. Par exemple dans le cas de TSS, chaque étape est considérée comme une recherche de 9 points. Dans le cas d'une recherche exhaustive de type FS, une seule phase est donc requise et la liste des adresses s'apparente à une simple incrémentation.

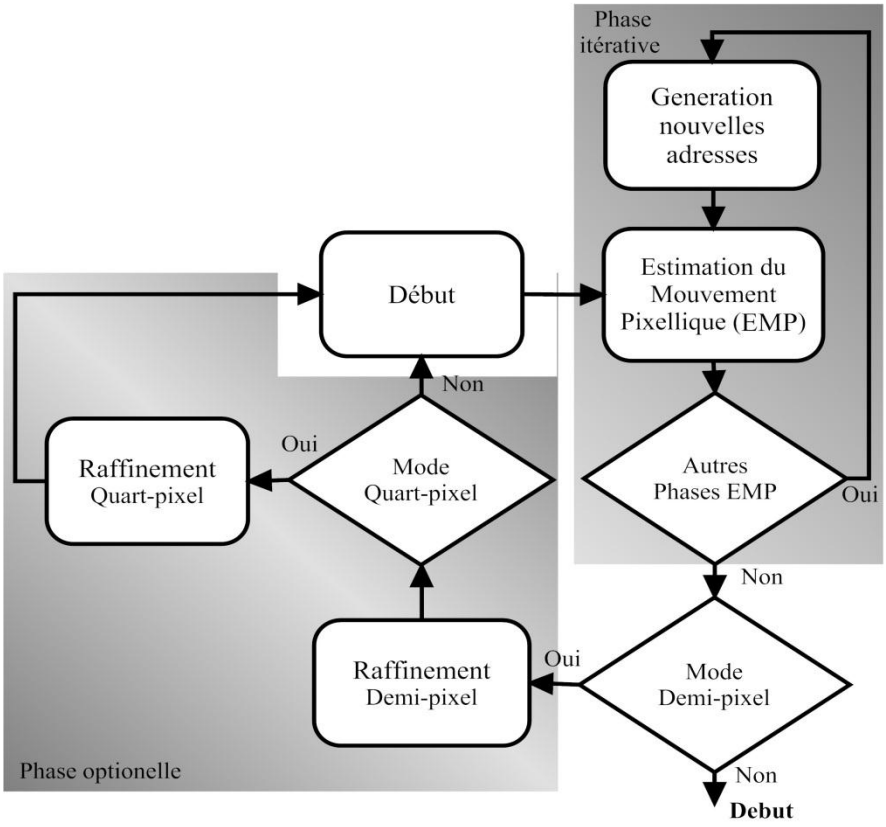


Figure 2. 21 : Flot algorithmique

La figure 2.21 est une description algorithmique de l'architecture configurable proposée. Cette architecture est basée sur deux phases principales : une phase itérative qui s'intéresse à l'estimation de mouvement pixélique et une phase optionnelle chargée de raffinement subpixélique. À partir de la figure 2.21 et selon l'étude ci-dessus, la première phase est réalisée à partir de trois modules. Le premier module se charge de la génération des adresses nécessaires pour une itération. Le deuxième effectue les calculs correspondant à chaque adresse, enfin le troisième module sert à contrôler l'état de la stratégie de recherche (convergence, réitération).

Selon la configuration sélectionnée, la phase optionnelle effectue un raffinement demi-pixel ou l'enchaînement des traitements demi puis quart de pixel.

L'architecture matérielle de l'estimateur configurable proposée est illustrée figure 2.22. Cette architecture est composée principalement par l'estimateur pixélique et l'estimateur subpixélique, une unité de stockage qui se charge de stocker le macro-bloc courant et la fenêtre de recherche, une unité d'extraction qui sert à sélectionner les pixels utiles et une unité externe de génération des adresses. Cette architecture sera détaillée dans le chapitre suivant.

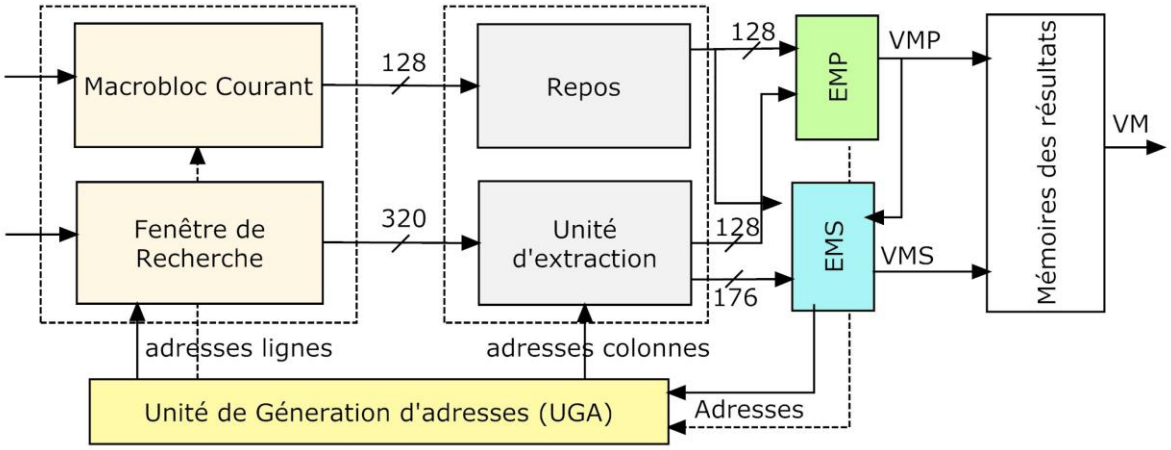


Figure 2. 22 : Architecture d'un estimateur de mouvement configurable

2.7. Conclusion

L'estimation de mouvement est une étape importante pour le traitement d'images et vidéo, et cette étape représente un véritable problème au niveau de la complexité de calcul. Nous avons rappelé et étudié les différentes techniques qui permettent l'estimation du mouvement. Parmi ces techniques, nous nous sommes intéressés seulement aux méthodes de type « Block Matching ». En effet, cette technique comporte une grande variété algorithmique permettant de chercher la similarité entre les blocs courant et référence. Nous avons pu relever que chaque stratégie possède ses propres avantages et inconvénients en termes de temps de traitement, complexité algorithmique et qualité d'image résultante.

Nous avons réalisé une étude analytique à partir des travaux antérieurs, et confirmé l'intérêt de définir un estimateur de mouvement configurable capable de supporter différentes stratégies de recherche, mais aussi les types de raffinements possibles. Ceci nous a conduits à la définition des principes généraux d'une architecture répondant aux contraintes citées, et nous allons maintenant détailler le rôle de chacun de ses modules, puis étudier ses performances et la comparer aux architectures existantes.

Chapitre III : Estimateur de mouvement : Implantation et résultats

3.1.Introduction

Comme nous l'avons précisé dans le chapitre précédent, l'ajustement des performances de l'estimation du mouvement peut se révéler primordial pour relever les challenges actuels et futurs dans le domaine de la transmission vidéo. Afin d'obtenir une grande variété de performances nous avons donc décidé d'agir sur les trois paramètres étudiés précédemment: le choix de **la stratégie de la recherche**, la nature des **tailles des blocs** (fixe ou variable) et la **précision d'estimation** (pixélique ou subpixélique) désirée. Nous avons donc défini une architecture de base pouvant être modifiée, voire s'ajuster en fonction de la configuration choisie aux niveaux de ces trois paramètres. Ces ajustements d'architectures seront plus ou moins importants en fonction de ces paramètres. L'objectif de ce chapitre est donc de décrire les différents éléments composant cette architecture, et d'évaluer leurs performances.

En premier lieu, l'estimation de mouvement est appliquée à la précision entière. En second lieu, un raffinement des vecteurs de mouvements obtenus durant la phase pixélique est préféré. L'amélioration de la qualité d'encodage observée après la phase de raffinement a, bien sûr, un coût en termes de volume de calculs. L'architecture matérielle devra donc privilégier la flexibilité au niveau de la phase pixélique afin de pouvoir adapter la stratégie de recherche aux besoins utilisateur et aussi exploiter au maximum le parallélisme potentiel de la phase de recherche subpixélique

La première partie de ce chapitre s'intéresse d'une part au premier élément configurable, c'est-à-dire la stratégie de recherche utilisée et d'autre part à l'intégration d'une recherche de type VBSME. La deuxième partie s'attache au raffinement des résultats obtenus, à la précision désirée : demi ou quart de pixel. L'architecture de base et ses variantes servent à l'optimisation de l'estimation de mouvement vis-à-vis de la complexité des calculs et des ressources matérielles nécessaires. Une discussion des résultats d'implantation et une comparaison avec des travaux antérieurs et récents sont finalement présentées.

3.2. Estimateur de mouvement pixélique

La phase de recherche pixélique est préalable à toute étape de raffinement de la recherche de mouvement. Le type de stratégie de recherche influe directement sur les performances de l'estimateur (qualité, temps de traitement). Dans le contexte de la définition d'une architecture configurable capable d'adapter ses performances de codage, cette dernière doit ainsi permettre de modifier l'algorithme définissant la recherche. Ce dernier doit alors être sélectionné en

fonction de l'application considérée ou encore des caractéristiques de la vidéo traitée (tels que les types des mouvements ou leur densité).

L'architecture matérielle de l'unité de traitement est un point clé dans l'estimation de mouvement pixélique. Comme nous l'avons décrit au chapitre précédent, de nombreuses architectures matérielles ont été proposées dans la littérature afin réaliser une estimation du mouvement avec une précision pixélique. Cependant ces architectures sont spécifiques et permettent soit une recherche exhaustive, soit une recherche de type rapide. Toutefois, à notre connaissance, aucune architecture unifiée supportant différentes stratégies de recherche n'a été encore étudiée. Dans le cadre de cette thèse, nous proposons une architecture matérielle configurable capable de supporter différentes stratégies de recherche en conservant l'intégralité de la partie opérative (permettant de réaliser les opérations de mise en correspondance de blocs). Seule la génération de l'enchaînement des différentes positions à examiner sera modifiée, en fonction de l'algorithme de recherche sélectionné. Cette partie correspond à un générateur d'adresses : celles des positions de blocs pour lesquels il est nécessaire d'évaluer la distorsion. De plus cette architecture devra supporter un traitement avec des blocs de tailles variables (mode VBSME). Dans le but de profiter de la rapidité du matériel d'une part et de la flexibilité du processeur d'autre part, afin d'augmenter les performances en temps réel, nous nous sommes intéressés à proposer une architecture matérielle à cible FPGA.

Afin d'optimiser les performances de l'architecture, la solution proposée sera développée en tenant compte de :

- L'accès parallèle aux données et l'organisation des mémoires de stockage;
- Le traitement séquentiel des macro-blocs;
- La propagation partielle et réutilisation de SAD;
- La réutilisation des unités de calcul.

3.2.1. Architecture matérielle de l'estimateur pixélique

La solution proposée doit pouvoir supporter la plupart des stratégies de recherche en particulier celle exhaustive. Les architectures les plus performantes pour ce type de recherche sont de type systolique. L'accélération du temps de calcul est obtenue par la mise en œuvre d'un grand nombre d'unités matérielles fonctionnant en parallèle. Bien entendu, le coût en termes de ressources matérielles est alors très important. Dans le cas d'algorithmes de recherche de type rapide, le nombre de positions à évaluer est alors drastiquement réduit

(potentiellement une réduction d'un facteur 100), l'architecture peut alors être plus économique en termes de ressources matérielles, et la recherche pixélique peut devenir plus rapide que le raffinement subpixélique. Par conséquent, les architectures systoliques, étant donné leur coût matériel exorbitant, apparaissent complètement inappropriées pour l'objectif définit. Nous proposons une solution avec un relativement faible coût matériel mais permettant cependant un traitement de mise en correspondance de blocs ligne par ligne. L'objectif est de pouvoir réaliser à chaque cycle d'horloge la comparaison d'une ligne du macro-bloc avec une portion équivalente extraite de la zone de recherche. Un des challenges est alors de pouvoir accéder aux 32 pixels nécessaires simultanément, à chaque cycle d'horloge, sans latence et quelle que soit la position considérée dans la zone de recherche. Une mémoire cache spécifique doit être élaborée pour assurer une telle fonctionnalité.

L'architecture matérielle proposée est une structure pipeline composée d'une unité de mémoire cache, d'une unité d'extraction des pixels, d'une unité de Processeurs Elémentaires (PE), une unité de comparaison et d'une unité de décision et d'une unité modifiable en charge de la génération des adresses. L'architecture de l'estimation est représentée figure 3.1. Les PE évaluent les distorsions pour chaque position candidate. L'opérateur de distorsion retenu est la SAD car elle nécessite beaucoup moins de ressources que les autres critères présentés dans le chapitre 1. La taille de bloc variable est supportée grâce à des paramètres de taille du bloc, modifiables dynamiquement. Dans la suite nous détaillerons les différentes unités de l'architecture globale.

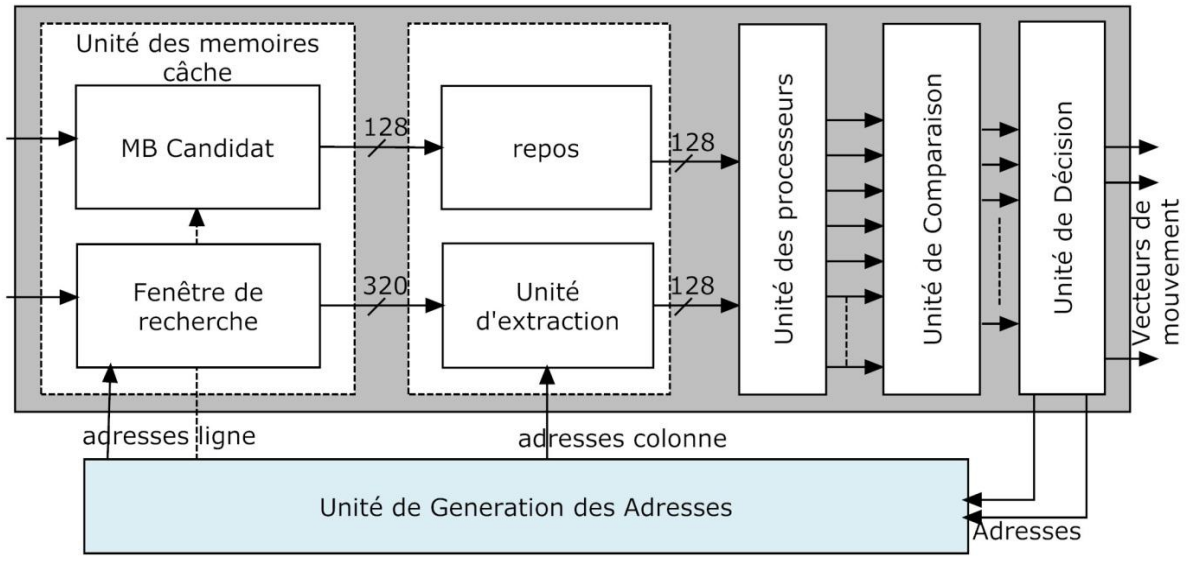


Figure 3. 1 : Schéma blocs de l'estimateur de mouvement pixélique

3.2.1.1. Unité des mémoires cache

Afin d'assurer une estimation de mouvement configurable capable de supporter différentes stratégies, nous avons proposé une nouvelle organisation des données dans les bancs mémoires. L'objectif principal de cette organisation est de permettre la parallélisation des opérations afin de réduire le temps de traitement. En outre, nous avons pris en considération la nécessité de minimiser les besoins en mémoire. Comme représenté dans la figure 3.1, l'unité des mémoires caches est composée de deux espaces mémoires indépendants pour le stockage des données du macro-bloc courant et de la fenêtre de recherche. Dans le but de minimiser les accès à la mémoire, et comme l'unité de traitement proposée est capable de calculer la SAD de 16 pixels en parallèle, l'idée principale est d'assurer l'émission de données en 16 cycles (une ligne de 16 pixels à chaque cycle).

La figure 3.2 représente l'architecture interne du module de mémoire cache dédié aux données de la fenêtre de recherche. Les blocs de contrôle (Ecriture, Lecture, Multiplexeur) sont décrits sous la forme des machines d'états. Ils permettent de générer les signaux de contrôle nécessaires pour assurer l'écriture et la lecture dans des BRAM cibles.

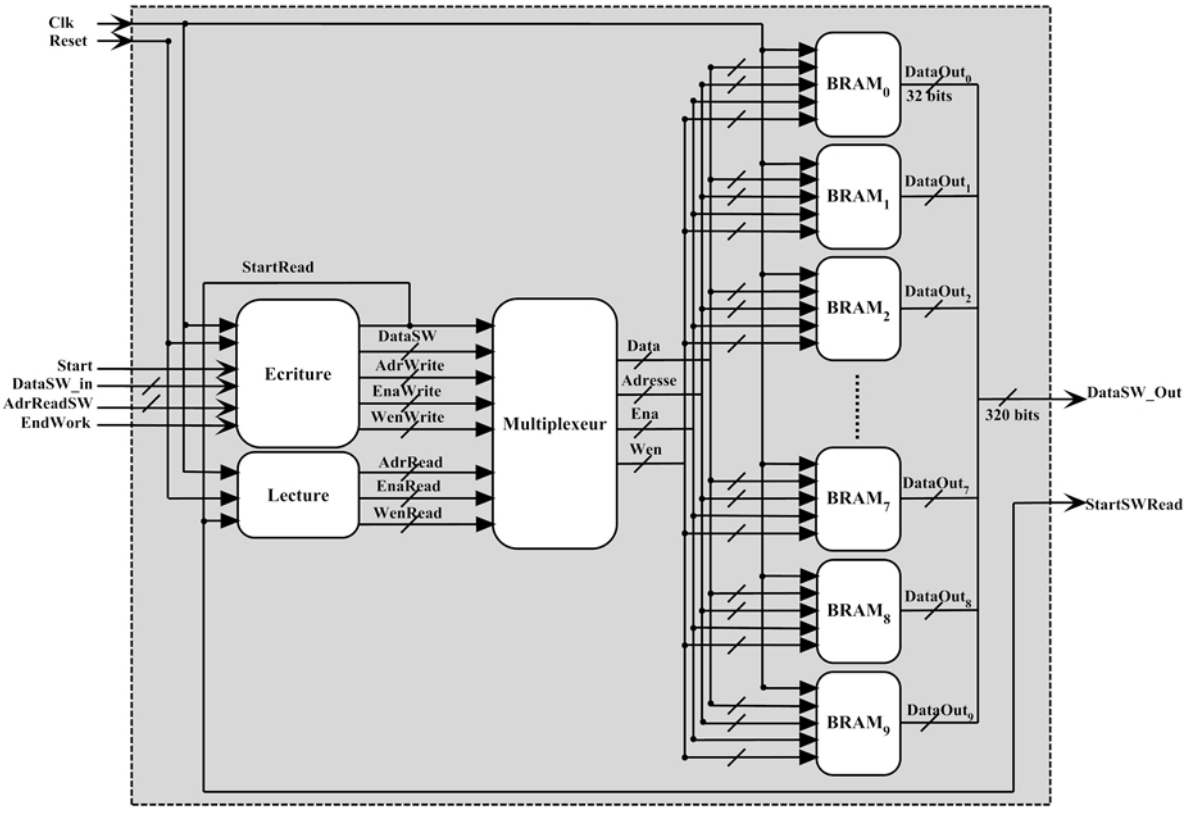


Figure 3. 2 : Architecture interne de gestion mémoire

Le composant multiplexeur permet d'une part d'aiguiller les données d'entrée vers les BRAMs concernées, et d'autre part, lors des phases de relecture des données de sélectionner une ligne de 40 pixels (320 bits) à chaque cycle d'horloge. La même architecture est utilisée avec les données du macro-bloc courant.

Rappelons que d'une manière générale, la fenêtre de recherche est un rectangle de taille NxM avec $\boldsymbol{N \leq M}$ ou inversement. Ce point de détail est utilisé pour l'optimisation des accès mémoire. Pour lire une ligne complète de N pixels, nous utilisons $\frac{\boldsymbol{N}}{\boldsymbol{4}}$ modules mémoire dupliquées parallèlement permettant le stockage d'un mot de 32 bits chacune comme représenté dans la figure 3.3. La duplication des modules mémoires en parallèle permet d'accéder aux différentes données sans conflit. Par ailleurs, dans le cas où $\boldsymbol{N > M}$, il est nécessaire de transposer à la volé les données pendant la phase de stockage afin de minimiser le nombre de modules mémoires, ce qui donne $\frac{\boldsymbol{M}}{\boldsymbol{4}}$ BRAM. Cette technique est utilisée dans ce travail lors de l'utilisation d'une fenêtre de recherche de taille 56x40. La transposition des pixels lors de stockage est indispensable afin de réduire le nombre des BRAM à 10 au lieu de 14. Ceci permet une réduction de la largeur de bus de sortie à 40x8bits au lieu de 56x8bits.

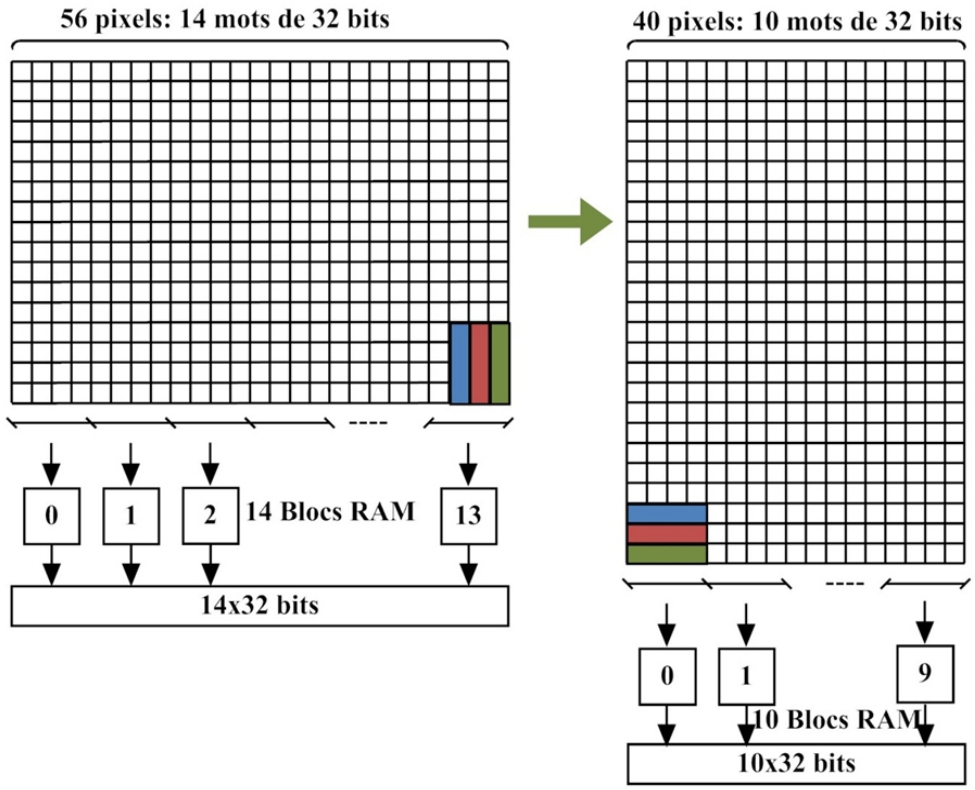


Figure 3. 3: Organisation de la mémoire contenant la fenêtre de recherche

3.2.1.2. Unité d'extraction des données

Toujours afin de minimiser l'accès à la mémoire et favoriser le traitement parallèle des données, une unité d'extraction des données utiles est proposée. La technique présentée dans notre travail consiste à transmettre à l'unité de traitement, une ligne de 16 pixels à chaque cycle. Les16 pixels peuvent alors être comparés en parallèle avec les 16 pixels de la ligne correspondant dans le macro-bloc courant. La manière de stocker les données influe sur le nombre des cycles nécessaires pour la lecture, comme expliqué précédemment.

Les données d'entrées arrivent donc sous forme de ligne de 16 pixels. Cette séquence est liée aux traitements amont des mémoires. Comme illustré dans la figure 3.4, cette unité a comme but d'extraire les 16 pixels de la ligne fournie par le module précèdent, en se basant sur l'adresse ***y*** du point à tester. Cette même unité est utilisée pour le traitement subpixélique. Le processus de lecture de la mémoire se fait donc en deux phases : une phase pour la lecture des BRAM et une seconde pour la sélection des pixels utiles.

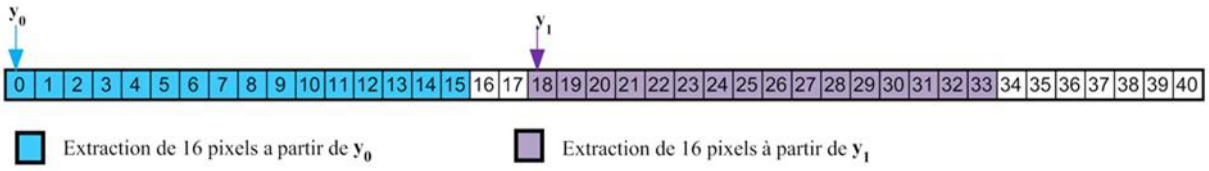


Figure 3. 4: Extraction des données utiles

3.2.1.3. Unité des processeurs

L'objectif principal de nos travaux est de proposer une architecture d'un estimateur de mouvement capable de supporter différentes stratégies en conservant l'intégralité de la partie opérative à savoir celle responsable du traitement d'un « matching ». Cette opération peut être réitérée pour chacune des positions à considérer du macro-bloc dans la fenêtre de recherche. Ces dernières étant favorables et définies par l'algorithme de recherche. Grâce aux unités de mémoires caches et celles d'extraction, mais aussi à cette stratégie consistant dans le traitement séquentiel des macro-blocs candidats, l'enchaînement entre deux matching peut s'effectuer sans aucune latence. Chaque « matching » à réaliser est défini par une adresse de base. Le changement d'algorithme est obtenu en modifiant le séquencement des adresses nécessaires pour chaque estimation.

Par ailleurs, l'architecture matérielle des processeurs élémentaires ainsi que leurs interconnexions représentent le cœur de l'estimation du mouvement pixélique et influent directement sur ses performances. Dans ce travail, nous avons proposé une structure de processeur qui supporte la taille variable de blocs basée sur la technique de la propagation

partielle du SAD Yap et al [YM04], Song et al [SLI+06], Fatemi et al [FAS09], Lee et al [LKK05]. C'est une architecture parallèle composée de six étages de pipeline. Comme illustré par la figure 3.5, Ces étages comportent 16 opérateurs de différence absolue situés au premier niveau de pipeline, 15 additionneurs distribués sur 4 étages, et un ensemble de 16 accumulateurs.

A chaque cycle d'horloge, une ligne de chaque macro-blocs courant et référence contenant 16 pixels, soit deux lignes au total, sont transmises aux 16 opérateurs de différences absolues afin de générer le résultat pour l'étage d'additionneurs qui suit. Six cycles sont nécessaires pour remplir les étages de pipeline, ce qui représente la latence de cette structure pipeline. Les accumulateurs connectés aux sorties des additionneurs permettent d'extraire en parallèle les valeurs de SAD pour les différentes tailles de sous blocs. D'autre part les résultats des SAD peuvent être réutilisés et accumulés pour calculer les autres valeurs de SAD pour les autres tailles des blocs. Par exemple, les valeurs SAD de deux blocs de tailles 4x4 peuvent être réutilisées pour obtenir les SAD correspondants aux blocs 4x8 ou 8x4, et ainsi de suite.

Les originalités de cette architecture sont:

- L'utilisation en parallèle des accumulateurs, ce qui permet de gagner un cycle d'horloge à chaque étage au cours du processus VBS,
- Les différentes correspondances peuvent être traitées d'une manière séquentielle, sans aucune latence.

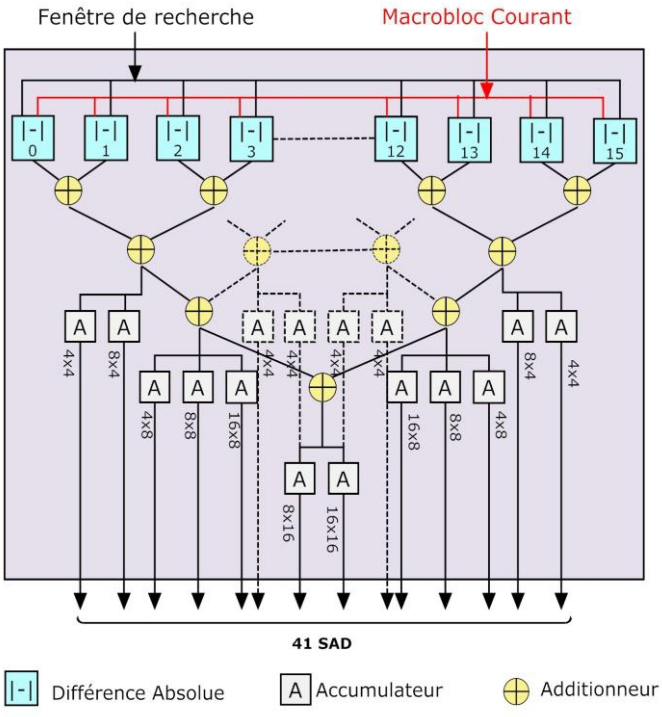


Figure 3. 5 : Unité des processeurs élémentaires à taille de blocs variable

3.2.1.4. Unité de comparaison

L'unité de comparaison sert à comparer les valeurs de SAD associées aux points testés afin de déterminer la valeur minimale de SAD à la fin de la recherche. Comme illustré figure 3.6, l'architecture interne de cette unité est composée de 41 comparateurs qui fonctionnent parallèlement assurant la comparaison pour chaque partition (4x4, 4x8, 8x4, 8x8, 8x16, 16x8, 16x16). La sortie de chaque unité est la valeur minimale de SAD.

Par la suite ces minima sont transmis à l'unité de décision qui se charge de renvoyer les résultats à l'unité de génération des adresses pour passer à l'étape suivante ou bien achever le calcul en fournissant les vecteurs mouvement, puis en passant au macro-bloc suivant de l'image courante.

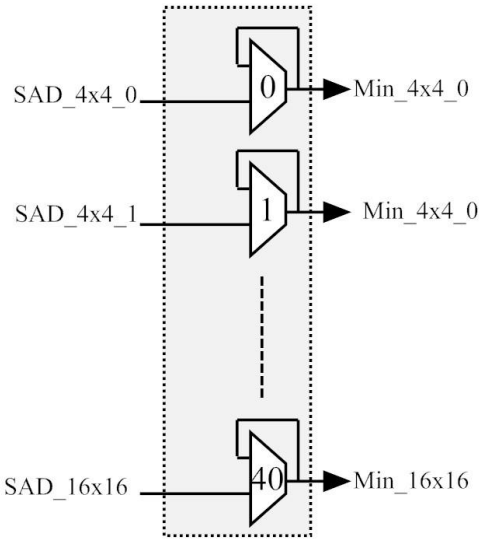


Figure 3. 6 : architecture interne d'unité des comparateurs

3.2.1.5. Unité de génération d'adresse : étude de cas FS & DS

Dans ce travail, deux algorithmes de types différents (FS et DS) ont été choisis pour valider la flexibilité de notre architecture. Pour FS toutes les adresses possibles sont déjà connues dès le début de la recherche, une simple incrémentation des adresses permet le balayage de la fenêtre de recherche. L'ensemble des adresses à estimer est ainsi prévisible et dépend de la région et la taille de recherche.

Les seules modifications à réaliser pour passer d'un algorithme à l'autre se trouvent au niveau de la partie génération des adresses. Ceci est notamment dû au traitement séquentiel des macro-blocs qui permet l'implantation des deux algorithmes de recherche tout en gardant la même architecture de traitement (partie opérative).

La figure 3.7 présente le séquencement des phases de génération des adresses pour l'algorithme de recherche en diamant "DS" et l'algorithme de recherche exhaustive "FS".

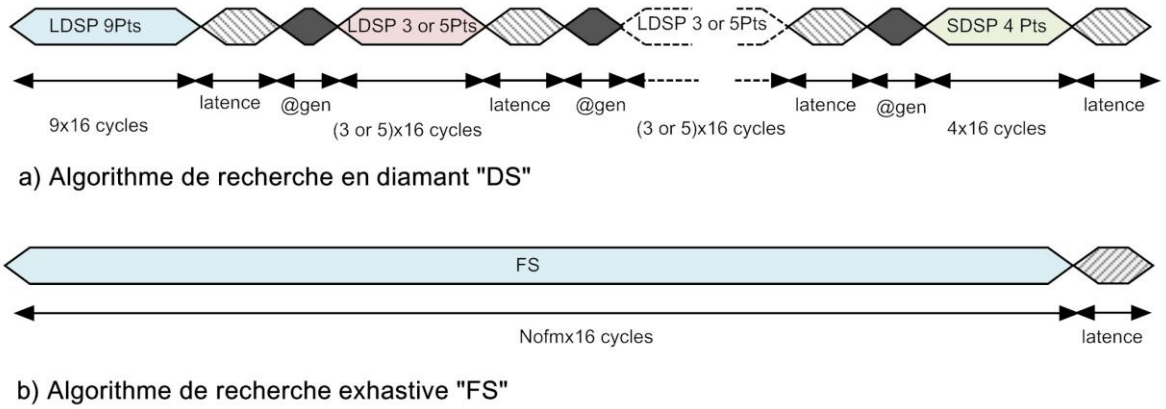


Figure 3. 7: Transfert des données et génération des adresses

Pour une fenêtre de recherche de taille NxM et un macro-bloc de taille PxP, le nombre maximal des positions à tester est (N-P+1)x(N-P+1) si l'algorithme de recherche choisi est de type exhaustif (FS). Par conséquent 16 adresses pour chaque macro-bloc seront envoyées aux blocs mémoires pour récupérer une ligne de N pixels à chaque cycle. L'unité d'extraction reçoit N pixels en entrée et suivant l'adresse reçue, sélectionne les 16 pixels utiles afin de les transmettre aux processeurs élémentaires. Le résultat final est obtenu six cycles après le transfert de la dernière ligne. Les traitements de deux macro-blocs associés à deux adresses aléatoires (non consécutives) peuvent être calculés sans aucune latence.

La recherche en grille de diamant comporte plusieurs phases pour la génération des adresses. Au début, le modèle large de la grille de diamant (LDSP) est appliqué, et nécessite le traitement de 9 macro-blocs comme indiqué dans l'algorithme approprié. Puis des séries de 3 ou 5 positions sont alors estimées en fonction du minimum de distorsion obtenu à la fin de la phase précédente d'estimation. Finalement, lors de l'ultime phase, 4 positions sont estimées. Ce séquencement d'adresses, irrégulier et permettant la succession de phases d'estimation, est commun à la plupart des estimations dites rapides. Il est de plus complètement différent de celui, très régulier, nécessaire à la stratégie FS. Cependant, l'adressage nécessaire à cette succession de phases peut être supporté sans problème par la partie opérative, qui peut calculer n'importe quel séquencement d'adresses, qui plus est sans aucune latence entre deux estimations. La seule latence existante est relative aux calculs des nouvelles positions à estimer lorsque la génération dépend des résultats d'estimation de la

phase antérieure. Dans le cas de l'algorithme DS, un délai fixe de 2 cycles est systématiquement nécessaire pour la génération des adresses utilisées par la prochaine phase de recherche.

3.2.2. Résultats d'implantation et discussion

De nos jours, les développeurs disposent de différentes méthodologies ou approches afin d'implanter leurs applications. En fonctions des contraintes de l'application, le concepteur définit son choix entre la conception d'une architecture figée ayant pour cible un ASIC (Application Specific Integrated Circuit) ou la famille des circuits reconfigurables FPGA (Field-Programmable Gate Array). Grâce à l'évolution de la technologie micro-électronique et des procédés de fabrication, l'utilisation des FPGA devient de plus en plus pertinente pour le prototypage rapide mais aussi pour le développement d'applications finales, étant donné ses grandes capacités d'intégration et de reconfiguration.

L'élément de base caractéristique de ce type de circuits est une cellule logique programmable, même si bien d'autres ressources internes sont venues progressivement se greffer sur cette base : multiplieurs câblés, blocs de mémoire, opérations de type multiplication-accumulation câblées dans des DSP Slices, émetteurs/récepteurs série rapides, etc. Depuis le début des années 2000, certains FPGA disposent même de cœurs de processeurs (PowerPC pour Xilinx et ARM pour Altera) qui favorisent la conception des systèmes mixtes tout en respectant les contraintes temps réel imposées par les applications, afin de bénéficier à la fois du faible coût et de la flexibilité de l'implantation en logiciel, mais aussi de la performance (rapidité d'exécution) de l'implantation en matériel.

Un FPGA est donc une matrice régulière de blocs logiques combinatoires et séquentiels placés dans une infrastructure d'interconnexions. Comme illustré dans la figure 3.8, un circuit FPGA est constitué d'un réseau de blocs logiques, de blocs mémoires, d'éléments de routages, d'éléments de contrôle des horloges, de blocs dédiés, et d'entrées/sorties. Les blocs logiques permettent de réaliser les opérations de logique combinatoire à base de LUT (Look Up Table) qui contiennent, après configuration, la table de vérité de la fonction logique qu'elles doivent réaliser. Ce principe reste généralement valable quels que soient les constructeurs. Les blocs RAM permettent notamment d'implanter des unités de mémoires adressables et des FIFO. Les blocs dédiés permettent de réaliser facilement de nombreuses opérations arithmétique (multiplieurs) et des opérations de traitement du signal (blocs DSP) ainsi que de commander

l'horloge. Le lecteur pourra se référer à [Bai11] pour une description plus complète des ressources internes de ces circuits.

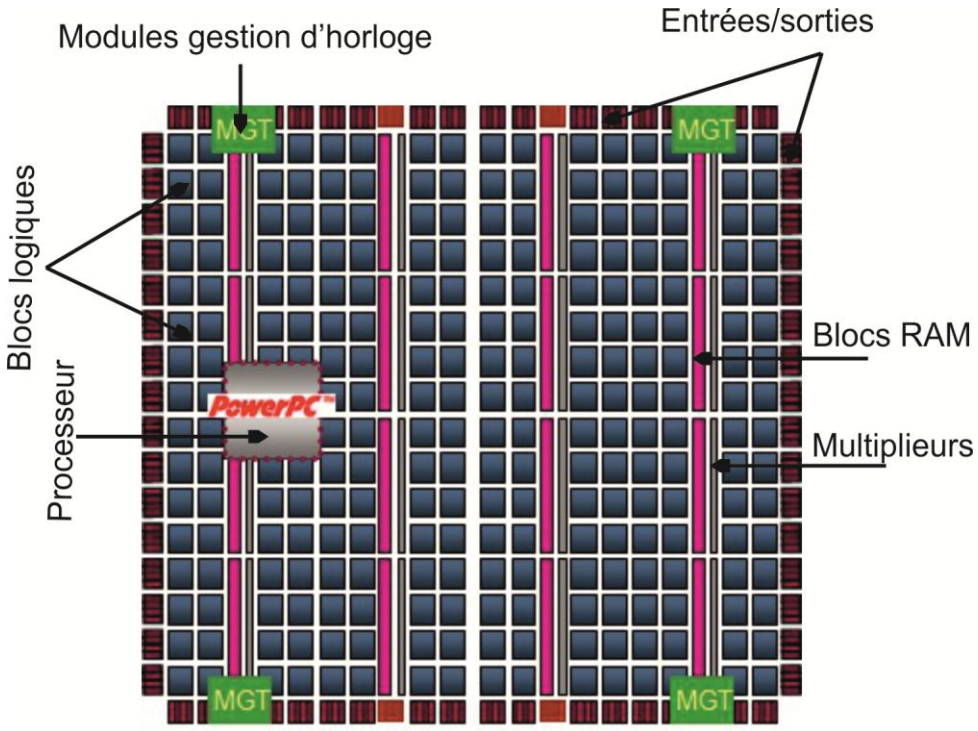


Figure 3. 8 : Structure interne d'un FPGA Xilinx de type Virtex II Pro

L'architecture de l'estimateur de mouvement à taille de bloc variable a été écrite en langage de description matérielle (VHDL), simulée, vérifiée et implantée sur un composant de type FPGA Virtex6 (6vlx240tff784-3). Les résultats d'implantation sont récapitulés dans la Table 3.1. Environ 1281 LUT (Look Up Tables) à six entrées sont nécessaires (1168 slices), ce qui représente un faible taux d'utilisation des ressources internes du composant (11%). Ceci permettra d'ajouter d'autres options et d'autres modules pour le codage vidéo. La fréquence maximale de fonctionnement est de 438 MHz.

Une étude comparative a été menée afin de positionner notre architecture vis-à-vis des architectures matérielles les plus performantes publiées précédemment. La comparaison porte sur la fréquence de fonctionnement, le temps nécessaire pour le traitement d'un macrobloc (latence), le nombre de PE utilisés, les stratégies supportées, ainsi que le débit maximum exprimé en Mblocks/s. Une synthèse de cette comparaison est reportée dans la table 3.2.

Notre architecture fait partie de la famille des implantations basées sur un faible nombre de PE (16), comme celles de Yap, Song ou encore Fatemi. Ceci permet d'économiser les

ressources matérielles, comparé aux architectures de Deng (256 PEs) ou Su-Jin (64 PEs), tout en assurant une latence du même ordre de grandeur.

Estimateur de Movement Pixélique (Device :6vlx240tff784-3)			
Ressources	**Utilisées**	**Disponibles**	**Occupation**
Nombre de Slices	1168	301440	<1%
Nombre de LUTs	1281	150720	<1%
Nombre de BRAMs	1	416	<1%
Fréquence Maximale	438MHz		

Table 3. 1 : Résultats de synthèse

Seule l'architecture de Su-Jin, dans sa version 16x16, permet d'avoir une latence significativement plus faible que la nôtre (1027 contre 4096) pour un débit légèrement supérieur (151.9 Mbloc/s contre 108.5), et ce à partir d'une fréquence de fonctionnement plus basse (178 MHz contre 438 MHz). Toutefois, cette architecture ne propose comme stratégie que la recherche exhaustive, ce qui ne lui permet pas de s'adapter de manière aussi souple que la nôtre, qui propose ici les stratégies FS et DS.

Ref.	nb PE's	searching range	Latency	Tech (µm)	Freq (MHz)	Max k Mblocks/s using FS	Max k Mblocks/s using DS
Yap	16	16x16	4096	0.13	294	72.9	n/a
Su-Jin	64	16x16	1027	Spartan 3 (0.9)	178	151.9	n/a
		32x32	4099			41.9	n/a
Deng	256	65x65	5216	0.18	260	48.6	n/a
Song	16	16x16	4096	0.18	266	9.9	n/a
Fatemi	16	32x32	26624	0.18	316	11.8	n/a
Ours	16	16x16	4096	Virtex 6 (0.04)	438	108.5	1806

Table 3. 2: Comparaison de différentes implantations

En résumé de cette analyse, nous pouvons donc conclure que notre architecture est compétitive avec les meilleures architectures d'estimation de mouvement du moment en termes de performances de traitement bruts, tout en apportant un plus via la possibilité pour l'utilisateur de choisir sa stratégie de recherche par simple modification du générateur d'adresses.

Par ailleurs, afin d'améliorer la qualité d'image dans les vidéos et d'augmenter les possibilités d'adaptabilité, l'utilisateur peut choisir la précision de raffinement désiré (demi ou quart de pixel). Nous allons donc maintenant décrire l'implantation de la partie subpixélique.

3.3.Estimateur de Mouvement Subpixélique

Durant la phase d'estimation de mouvement pixélique, 41 vecteurs de mouvement sont obtenus pour chaque type de bloc (4x4, 4x8, 8x4, 8x8, 8x16, 16x8, 16x16). Le passage à l'estimation de mouvement subpixélique exige un raffinement pour chaque type de précision (demi et quart de pixel). Ce raffinement est assuré à l'aide d'une étape d'interpolation de la zone de recherche. Cette interpolation consiste à calculer la valeur d'un pixel imaginaire situé entre deux pixels adjacents. De nombreux travaux ont été proposés dans le but de déterminer la bonne valeur. Dans la norme H.264/AVC, les demi-pixels d'un bloc sont interpolés à base de filtres à réponse impulsionnelle finie à six coefficients. Les quarts de pixels seront obtenus par filtrages bilinéaires appliqués aux demi-pixels calculés précédemment.

Dans ce contexte, le compromis entre le temps d'exécution, l'accès à la mémoire et le taux d'utilisation des ressources matérielles doit être équilibré afin d'optimiser les performances de l'estimateur. Pour cela, une combinaison de différentes architectures est présentée, qui offre la possibilité d'ajuster le codeur selon les besoins applicatifs. L'idée est d'introduire le traitement pipeline lors de deux raffinements et/ou augmenter le degré de parallélisme.

3.3.1. Architecture séquentielle et Architecture pipeline

Dans ce travail, nous avons proposé deux nouvelles architectures d'un estimateur de mouvement subpixélique pour atteindre une précision quart de pixel.

La première architecture est caractérisée par le traitement séquentiel du demi et quart de pixel, la deuxième exploite un traitement en pipeline. Cette dernière est plus rapide mais plus coûteuse en ressources matérielles par rapport à la première.

Les deux architectures utilisent les mêmes modules principaux (les unités d'interpolation, les unités des processeurs, les unités de comparaison, etc). Cependant, le séquencement de traitement et l'ordonnancement des tâches diffèrent entre les deux modèles.

- ***Architecture séquentielle***

Comme illustré figure 3.9, l'architecture séquentielle proposée est composée de trois principaux modules : le module « Demi-Pixel » encadré en vert, le module « Quart-Pixel » en bleu et le module de traitement en jaune. Les unités demi et quart de pixel permettent l'interpolation pour chaque niveau de raffinement afin de stocker les pixels utiles dans les bancs mémoire appropriés. La liaison avec le module de traitement est réalisée à travers un routeur qui se charge de sélectionner les données valides afin de les transmettre vers l'unité de traitement. Cette dernière contient une matrice de processeurs, en charge des calculs de distorsion pour chaque candidat, mais aussi deux comparateurs et un arbre de décision.

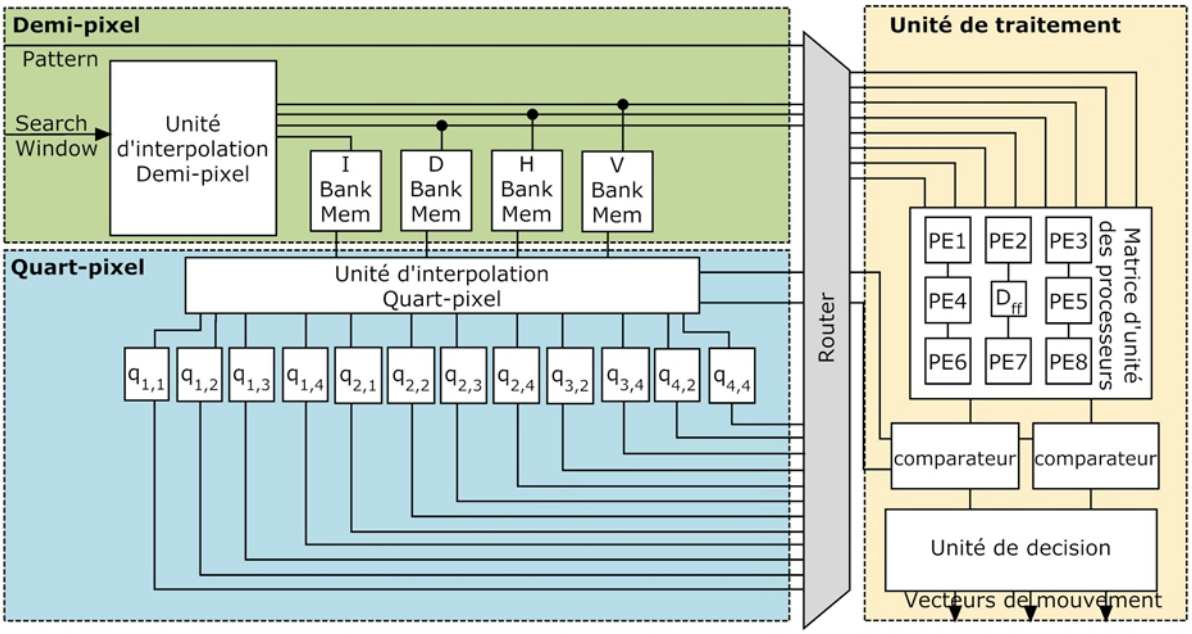


Figure 3. 9: Architecture séquentielle de l'estimation de mouvement subpixélique

Cette architecture est séquentielle étant donné que les deux processeurs "Demi" et "Quart" fonctionnent l'un après l'autre, ce qui permet d'avoir une unité de traitement unique, commune pour les deux raffinements. Cette solution est économique, mais elle est lente pour les applications utilisant des algorithmes de recherches rapides. En effet dans ce cas la phase subpixélique peut prendre deux fois plus de temps que la phase pixélique ce qui entraine un

déséquilibre au niveau de l'enchaînement en pipeline des phases pixéliques et subpixéliques. L'accélération de la phase subpixélique apparaît indispensable afin d'équilibrer les tâches. Ceci peut être réalisé grâce à une architecture introduisant un traitement pipeline.

- ***Architecture pipeline***

L'architecture pipeline est schématisée dans la figure 3.10. Elle est composée de deux processeurs qui fonctionnent en mode pipeline afin de paralléliser les calculs pour le demi et le quart de pixel. Chaque processeur se compose d'une unité principale d'interpolation, d'une unité de processeurs élémentaires, d'une unité de comparateurs et d'une unité de stockage. Toutefois, la présence d'une seule unité de traitement n'est pas suffisante pour assurer le pipeline désiré. Nous avons donc ajouté une unité de traitement supplémentaire par rapport à la solution séquentielle. Cette unité sera dédiée au traitement quart de pixel.

Le traitement en pipeline de deux raffinements demi et quart de pixel accélère logiquement la phase subpixélique, et ce d'un facteur qui peut aller jusqu'à deux, par rapport à l'architecture séquentielle. Cependant, un coût important en surface de silicium ou ressources matérielles est ajouté, ce qui conduit également à une consommation plus importante.

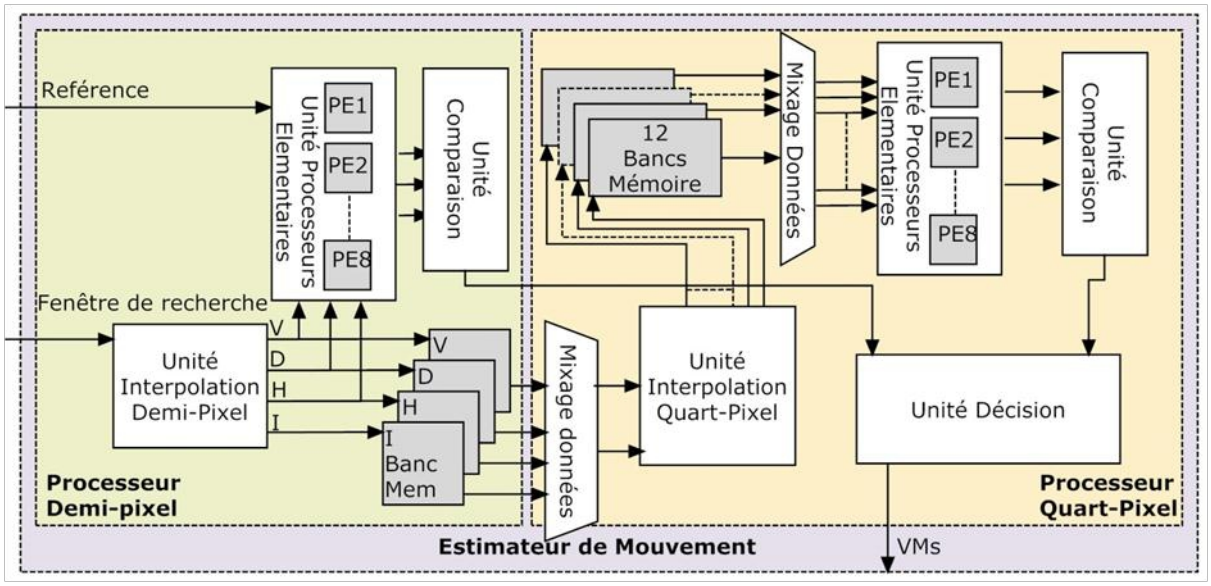


Figure 3. 10: Architecture pipeline d'estimation de mouvement subpixélique

Les deux solutions proposées sont essentiellement basées sur les mêmes modules que nous allons maintenant détailler ainsi que l'organisation et l'ordonnancement du flot de données qui se révèle nécessairement fondamentaux pour la mise en œuvre de ces architectures.

3.3.1.1. Unité d'interpolation

Dans la norme H.264/AVC, l'interpolation est réalisée en deux étapes de filtrage. L'interpolation demi-pixel est effectuée par le filtre à réponse impulsionnelle finie (FIR) à six coefficients (1, -5, 20, 20, -5, 1) comme indiqué dans la figure 3.11. Chaque demi-pixel est obtenu par l'interpolation de six pixels entiers. Deux types d'interpolation sont nécessaires : horizontale mais aussi verticale. Par exemple, pour calculer le demi-pixel $v_{3,3}$, les six pixels verticaux ($I_{3,1}$, $I_{3,2}$, $I_{3,3}$, $I_{3,4}$, $I_{3,5}$ et $I_{3,6}$) sont utilisés.

$$v_{3,3} = round\ ((I_{3,1} - 5\ I_{3,2} + 20\ I_{3,3} + 20\ I_{3,4} - 5\ I_{3,5} + I_{3,6})/32)$$

Il est également possible, pour calculer $h_{3,3}$ d'utiliser les six pixels horizontaux ($I_{1,3}$, $I_{2,3}$, $I_{3,3}$, $I_{4,3}$, $I_{5,3}$ et $I_{6,3}$) tels que :

$$h_{3,3} = round\ ((I_{1,3} - 5\ I_{2,3} + 20\ I_{3,3} + 20\ I_{4,3} - 5\ I_{5,3} + I_{6,3})/32)$$

L'interpolation réalisée avec ce type de filtre est relativement complexe, mais en contrepartie donne une bonne précision.

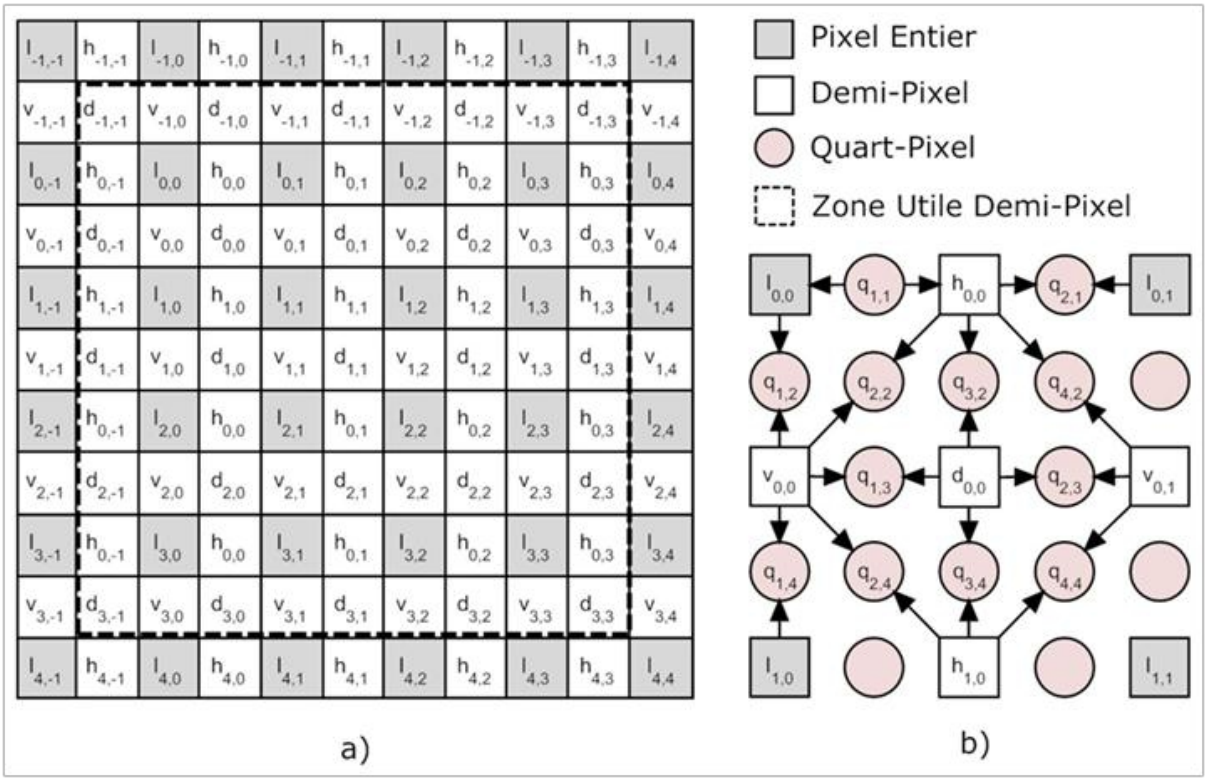


Figure 3. 11: a) Interpolation demi-pixel, b) Interpolation quart-pixel

Les quart-pixels sont calculés en utilisant des filtres bilinéaires. Ce type de filtre consiste à calculer la moyenne entre deux pixels adjacents. Il est plus simple à implanter que le filtre à six coefficients. Sur la figure 3.11 (b), chaque pixel situé à la connexion de deux flèches de

sens opposés indique un quart de pixel calculé à partir des demi-pixels ou des pixels entiers. Il existe 12 types de quart-pixels suivant 12 directions différentes. Le quart-pixel $q_{1,1}$ est calculé comme suit :

$$q_{1,1} = round((i_{1,1} + h_{1,1})/2)$$

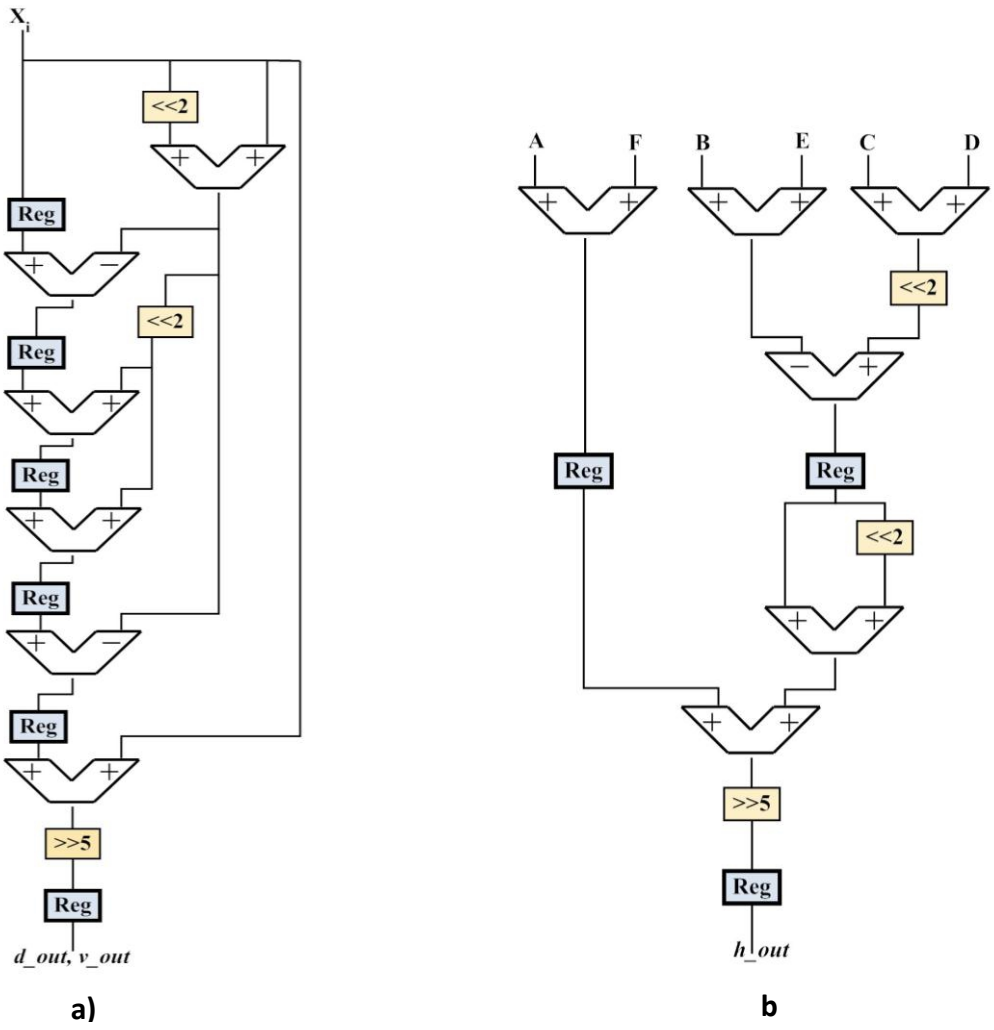


Figure 3. 12: Architecture interne des filtres FIR, a) FIR verticale, b) FIR horizontale

Dans les travaux présentés dans [CHC04][YGI06], différentes manières d'implanter les filtres FIR ont été proposées, sur des cibles de type ASIC. Nous nous sommes inspirés de ces implantations pour les architectures des filtres verticaux et horizontaux. La figure 3.12 montre l'architecture interne des filtres FIR à six coefficients FIR-H et FIR-V conformes à la norme H.264/AVC. Ces deux types de filtres (vertical et horizontal) sont composés de registres à décalage utilisés pour la multiplication ou la division, d'opérateurs d'addition ou de

soustraction et de registres internes qui définissent les étages pipelines. Le filtre FIR-V présente une latence de 6 cycles, contre 2 pour le filtre FIR-H.

Pour expliquer le processus d'interpolation demi-pixel, nous avons représenté figure 3.13 l'architecture interne de l'unité d'interpolation permettant de traiter un macro-bloc de taille 4x4.

Pour assurer ce calcul cette unité nécessite une ligne de 10 pixels entiers à chaque cycle et durant 10 cycles. Les nouveaux pixels générés et ceux entiers d'origine, se propagent à travers les registres pour conserver la notion de pipeline. Les filtres **h** sont les filtres à six coefficients horizontaux FIR-H, les filtres **v** sont les filtres verticaux de type FIR-V déjà mentionnés (figure 3.12).

Au total (voir figure 3.12), l'unité d'interpolation est composée de 5 filtres **h** et 11 filtres **v**. Six pixels {A, B, C, D, E, F} parmi les 10 pixels d'entrées sont connectés aux filtres **h**. Les six pixels sont également connectés aux registres intermédiaires (en gris sur la figure 3.13). Ces derniers permettent la propagation des pixels afin que les filtres verticaux puissent calculer les demi-pixels correspondants.

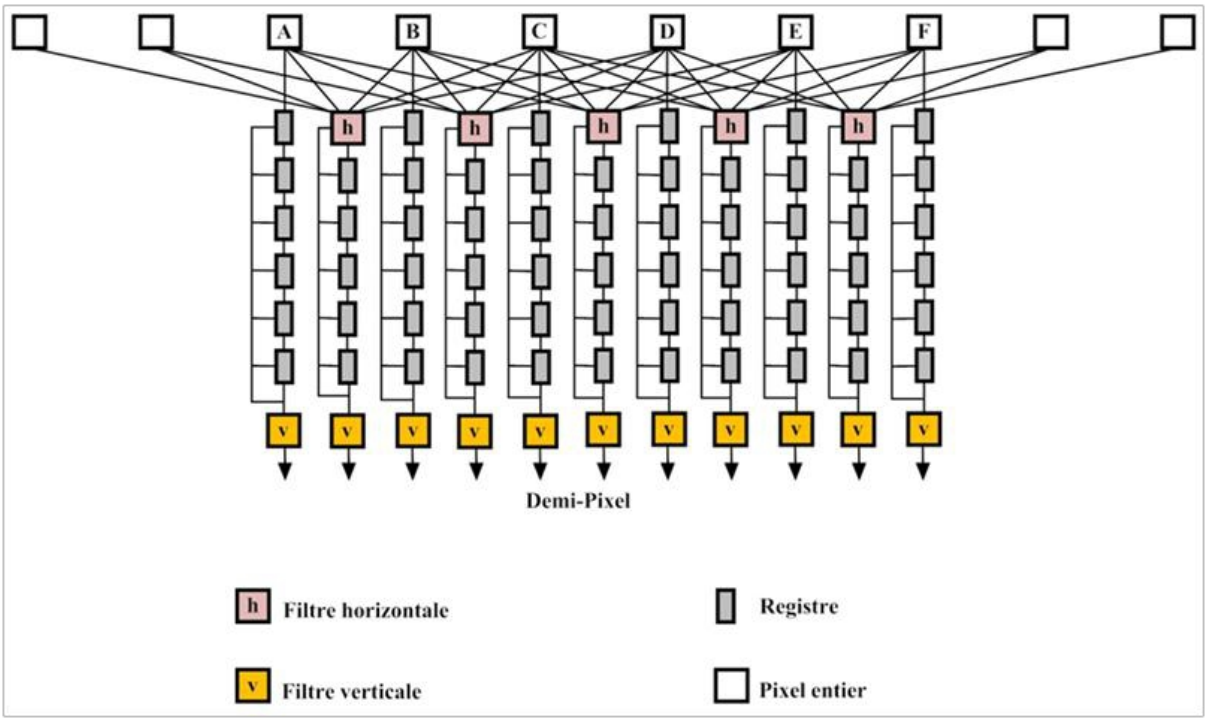


Figure 3. 13: Architecture d'unité d'interpolation du demi-pixel

La figure 3.14 représente l'architecture interne de l'unité d'interpolation concernant le raffinement quart de pixel. Cette unité est basée sur le filtre bilinéaire "B" qui consiste à

calculer la moyenne de deux pixels adjacents dans des orientations conformes à la norme H.264.

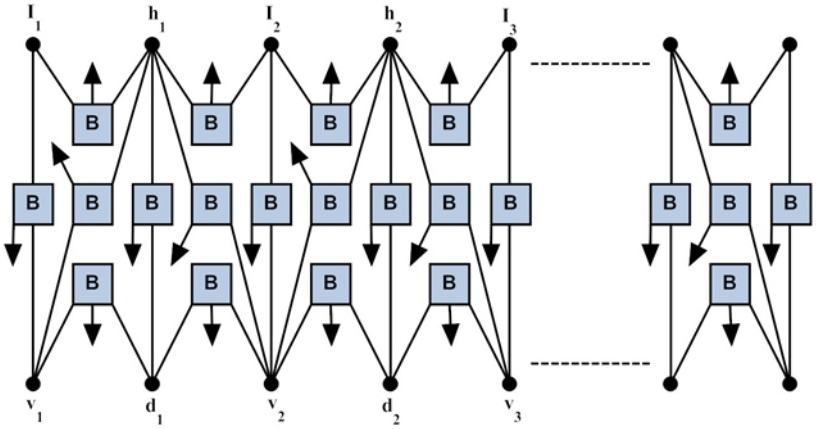


Figure 3. 14 : Architecture de l'unité d'interpolation quart-pixel

Comme montré sur la figure 3.14, cette unité est composée de trois lignes de filtres bilinéaires. La première ligne est chargée de calcul des quart-pixels "hi", tandis qu'une deuxième ligne s'intéresse aux quart-pixels "vd". La troisième ligne des filtres calcule les quart-pixels verticaux et diagonaux.

À chaque cycle, deux lignes de différents types "hi" et "vd" sont nécessaires pour calculer les coefficients quart de pixels qui seront stockés par la suite dans les mémoires cibles.

3.3.1.2. Gestion de la mémoire et ordonnancement du flux de données

La relation « Organisation de la mémoire/Interconnexions des unités de traitement » joue toujours un rôle très important dans l'optimisation des performances des systèmes électroniques. Ces deux paramètres influent directement sur la phase d'estimation de mouvement, ce qui oriente les chercheurs à tenter de minimiser les accès mémoires pour optimiser le temps de traitement.

Afin d'obtenir un bon compromis entre la complexité de calcul et complexité de la gestion de la mémoire lors de l'interpolation des demi-pixels, nous avons décidé de stocker des valeurs dans les mémoires réservées pour qu'elles soient prêtes à être utilisées lors de l'interpolation quart-pixel. Ainsi, les valeurs de pixels entiers {I} ainsi que celles des demi-pixels générées par l'unité de l'interpolation {H}, {V} et {D} sont stockées dans une banque

de quatre DPRAM. En effet, pendant la phase de raffinement demi-pixel les données sont transmises vers le routeur sans nécessité de stockage. Cependant, pour passer à la précision quart-pixel, le stockage des pixels interpolés et entiers est indispensable. Pour cela, nous avons proposé 4 plans de stockage selon la nature de pixel (I, H, V ou D) pour simplifier l'accès aux mémoires (il est possible de lire une ligne complète en un cycle). L'utilisation des blocs de mémoire en Dual-Port RAM (DPRAM) favorise la possibilité de mise en œuvre d'un traitement pipeline entre demi et quart de pixel. Ces plans mémoires sont utilisés pour stoker les pixels utiles pour toutes les tailles des blocs. Bien que traitant tous les sous blocs, la capacité mémoire doit être déterminée de manière à permettre le traitement du bloc de taille maximale, à savoir 16x16. Puisque l'unité d'interpolation quart de pixel nécessite deux types de donnes "hi" et "vd", les pixels sont stockés ligne par ligne en utilisant des mots de (n+2) pixels pour {I} et {V} et des mots de (n+1) pixels pour {H} et {D}, (n= 4, 8, 16).

Comme montré dans la figure 3.15, il existe 12 types de quart de pixels ($\mathbf{q_{1,1}}$, $\mathbf{q_{1,2}}$, $\mathbf{q_{1,3}}$, $\mathbf{q_{1,4}}$, $\mathbf{q_{2,1}}$, $\mathbf{q_{2,2}}$, $\mathbf{q_{2,3}}$, $\mathbf{q_{2,4}}$, $\mathbf{q_{3,2}}$, $\mathbf{q_{3,4}}$, $\mathbf{q_{4,2}}$, $\mathbf{q_{4,4}}$), chacun d'eux étant représenté avec une couleur spécifique sur cette figure. Ce partitionnement est très utile pour l'organisation des mémoires ainsi que l'ordonnancement de l'interpolation, ce qui explique l'utilisation des 12 DPRAM pour le stockage des données quart de pixel.

Supposons par exemple que $\mathbf{I_{1,1}}$ est le vecteur de mouvement obtenu durant l'estimation de mouvement pixélique. Le raffinement demi-pixel pourrait avoir comme solution un des huit candidats autour du centre, soit ($\mathbf{d_{0,0}}$; $\mathbf{d_{0,1}}$; $\mathbf{d_{1,0}}$; $\mathbf{d_{1,1}}$; $\mathbf{h_{1,0}}$; $\mathbf{h_{1,1}}$; $\mathbf{v_{0,1}}$; $\mathbf{v_{1,1}}$) .

La manière d'interpoler et de stocker les données n'est donc pas la même pour toutes les possibilités de demi-pixels, ce qui implique une complexité certaine. Une solution simple consisterait à traiter chaque possibilité individuellement, mais cette solution serait trop coûteuse en surface de silicium.

Nous proposons une solution générique, quelle que soit la position du demi-pixel obtenu, basée sur un contrôleur (routeur) se chargeant de l'extraction des données demi-pixels des mémoires {I, H, V, D}, afin de stoker les quart-pixels valides générés dans les huit mémoires appropriées.

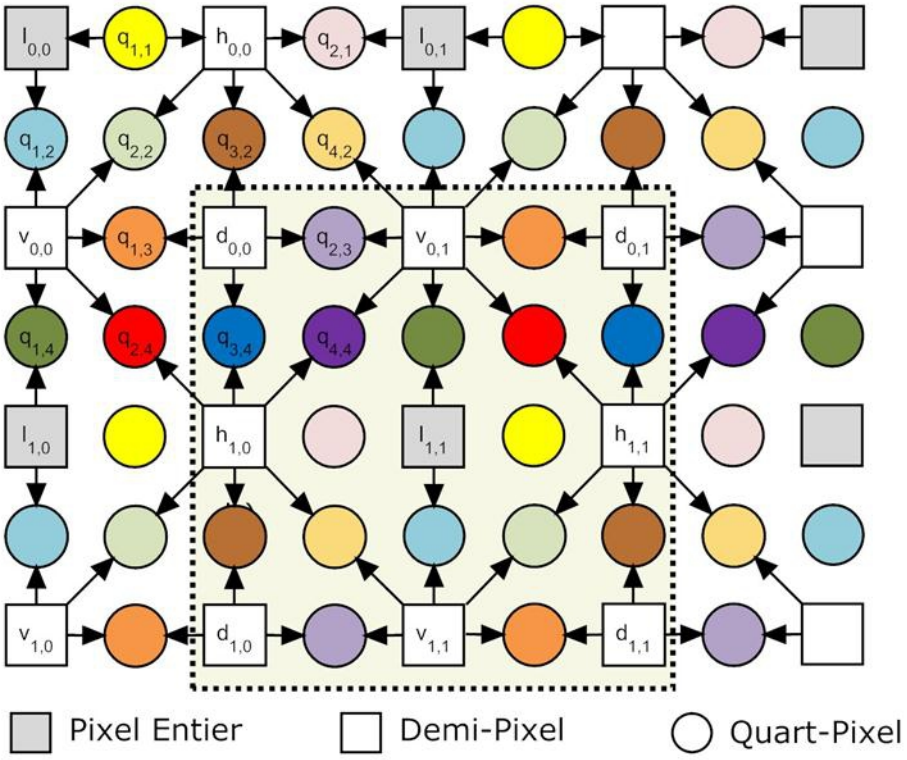


Figure 3. 15 : Différents types quart-pixel

Les données d'entrées pour les processeurs changent selon le type de raffinement (demi ou quart). Une ou deux unités de routage, respectivement pour l'architecture pipeline et celle séquentielle sont respectivement nécessaires pour garantir le bon acheminement des pixels. En effet, dans le cas de l'architecteur séquentielle, les deux unités de mixage des données utilisées au sein de l'architecture pipeline (figure 3.10), sont fusionnées en un seul composant de routage (nommé routeur) comme illustré figure 3.9. Le routeur ainsi obtenu permet de multiplexer les deux flots de données vers la matrice unique de PE présente dans le modèle séquentielle. Les deux unités de mixage respectivement utiles pour les raffinements demi et quart de pixels vont être maintenant décrites.

A l'issue de l'interpolation demi-pixélique, le flux des pixels interpolés est transmis vers le routeur qui se charge de sélectionner les pixels utiles afin de les transmettre vers la matrice des unités de processeurs. Pour la phase de raffinement subpixélique, le routeur sélectionne et réorganise les demi-pixels {H, V, D}. Les processeurs élémentaires sont activés avec un décalage d'un cycle (figure 3.16).

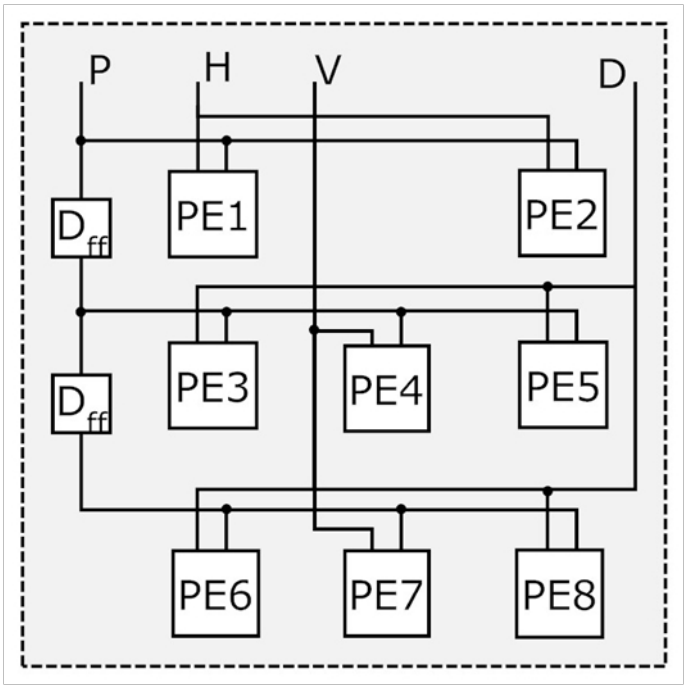


Figure 3. 16 : Interconnexion des PE / BRAM

De la même manière, durant la phase de raffinement quart de pixel, selon la position du vecteur de mouvement obtenu en demi-pixel, 8 blocs de RAM parmi 12 seront sélectionnées. Chaque mémoire sera connectée à un processeur élémentaire. L'unité de mixage des données ou de routage sert à sélectionner 8 DPRAM parmi 12 ($\mathbf{q_{1,1}}$, $\mathbf{q_{1,2}}$, $\mathbf{q_{1,3}}$, $\mathbf{q_{1,4}}$, $\mathbf{q_{2,1}}$, $\mathbf{q_{2,2}}$, $\mathbf{q_{2,3}}$, $\mathbf{q_{2,4}}$, $\mathbf{q_{3,2}}$, $\mathbf{q_{3,4}}$, $\mathbf{q_{4,2}}$, $\mathbf{q_{4,4}}$) selon la position du meilleur score obtenu lors de raffinement demi-pixel. L'architecture interne du routeur pour la phase quart-pixel est illustrée par la figure 3.17, où huit multiplexeurs à trois entrées sont interconnectés directement avec les 12 DPRAM afin de sélectionner la sortie qui sera utilisée par chaque processeur.

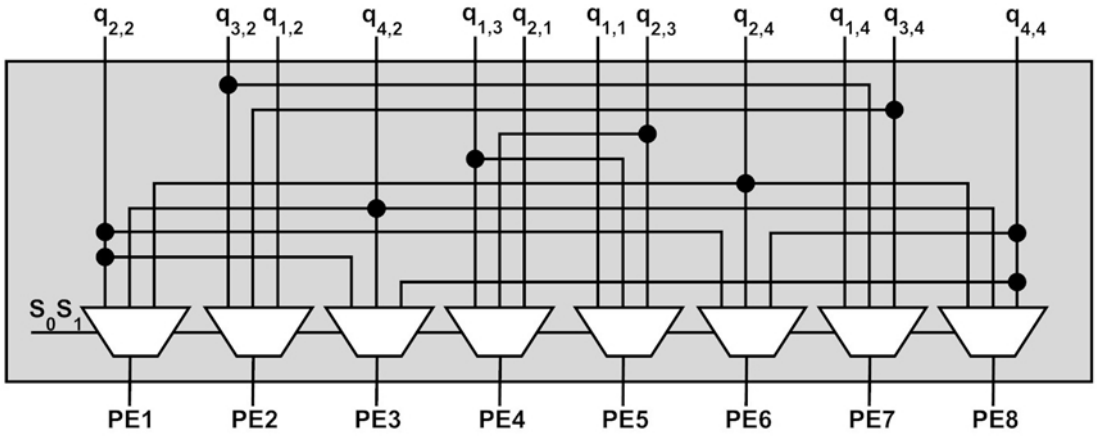


Figure 3. 17 : Routeur quart-pixel

Le tableau 3.3 illustre l'ordonnancement du flux des données durant la phase de raffinement demi-pixel. Pour des raisons de simplicité, le séquencement présenté est restreint au cas d'un macro-bloc de taille 4x4.

Pour chaque macro-bloc de taille 4x4, 5x4 demi-pixels horizontaux {h}, 4x5 demi-pixels verticaux {v} et 5x5 demi-pixels diagonaux {d} sont calculés. Les pixels interpolés sont transmis ligne par ligne chaque cycle d'horloge. Les pixels de types **h** sont disponibles à la sortie un cycle après les pixels de type **v** et **d**. Une synchronisation au niveau de la réception (unité de traitement) des pixels est indispensable. Comme déjà représenté dans la figure 3.16, les processeurs élémentaires sont activés les uns après les autres avec un retard d'un cycle. Au premier cycle, les pixels du macro-bloc courant sont transmis parallèlement vers PE1 et PE2.

La propagation vers les autres processeurs est assurée par des registres $\mathbf{D_{ff}}$ qui permettent de retarder les données d'un cycle d'horloge. Lors de la diffusion des demi-pixels, une extraction de 4 pixels parmi 5 est nécessaire pour les pixels de type **h**. Pour les pixels de type **v,** une extraction de deux ensembles de 4 parmi 5 pixels sont requis. Cet ordonnancement est traduit dans la figure 3.16, ce qui explique également le rôle de routeur d'acheminement.

Cycle	P	H	D	V	PE1	PE2	PE3	PE4	PE5	PE6	PE7	PE8	SAD
0	$p_{0,j}$	$h_{0,j-1}$ $h_{0,j}$			$\lvert h_{0,j-1}-p_{0,j}\rvert$	$\lvert h_{0,j}-p_{0,j}\rvert$	----	----	----	**Previous block**			
1	$p_{1,j}$	$h_{1,j-1}$ $h_{1,j}$	$d_{-1,j-1}$ $d_{-1,j}$	$V_{-1,j}$	$\lvert h_{1,j-1}-p_{1,j}\rvert$	$\lvert h_{1,j}-p_{1,j}\rvert$	$\lvert d_{-1,j-1}-p_{0,j}\rvert$	$\lvert v_{-1,j}-p_{0,j}\rvert$	$\lvert d_{-1,j}-p_{0,j}\rvert$	----	----	----	
2	$p_{2,j}$	$h_{2,j-1}$ $h_{2,j}$	$d_{0,j-1}$ $d_{0,j}$	$V_{0,j}$	$\lvert h_{2,j-1}-p_{2,j}\rvert$	$\lvert h_{2,j}-p_{2,j}\rvert$	$\lvert d_{0,j-1}-p_{1,j}\rvert$	$\lvert v_{0,j}-p_{1,j}\rvert$	$\lvert d_{0,j}-p_{1,j}\rvert$	$\lvert d_{0,j-1}-p_{0,j}\rvert$	$\lvert v_{0,j}-p_{0,j}\rvert$	$\lvert d_{0,j}-p_{0,j}\rvert$	
3	$p_{3,j}$	$h_{3,j-1}$ $h_{3,j}$	$d_{1,j-1}$ $d_{1,j}$	$V_{1,j}$	$\lvert h_{3,j-1}-p_{3,j}\rvert$	$\lvert h_{3,j}-p_{3,j}\rvert$	$\lvert d_{1,j-1}-p_{2,j}\rvert$	$\lvert v_{1,j}-p_{2,j}\rvert$	$\lvert d_{1,j}-p_{2,j}\rvert$	$\lvert d_{1,j-1}-p_{1,j}\rvert$	$\lvert v_{1,j}-p_{1,j}\rvert$	$\lvert d_{1,j}-p_{1,j}\rvert$	----
4	----	----	$d_{2,j-1}$ $d_{2,j}$	$V_{2,j}$	----	----	$\lvert d_{2,j-1}-p_{3,j}\rvert$	$\lvert v_{2,j}-p_{3,j}\rvert$	$\lvert d_{2,j}-p_{3,j}\rvert$	$\lvert d_{2,j-1}-p_{2,j}\rvert$	$\lvert v_{2,j}-p_{2,j}\rvert$	$\lvert d_{2,j}-p_{2,j}\rvert$	----
5			$d_{3,j-1}$ $d_{3,j}$	$V_{3,j}$			----	----	----	$\lvert d_{3,j-1}-p_{3,j}\rvert$	$\lvert v_{3,j}-p_{3,j}\rvert$	$\lvert d_{3,j}-p_{3,j}\rvert$	PE1 PE2
6					**Next block**					----	----	----	PE3 PE4 PE5
7													PE6 PE7 PE8
8													----

Table 3. 3 : Stockage et diffusion des données

De même qu'à l'émission des données, le décalage au niveau de la sortie des valeurs du SAD sera conservé par propagation. Les PEs sont ainsi activés par ensemble des dépendances de données {**PE$_1$**, **PE$_2$**}, {**PE3**, **PE$_4$**, **PE$_5$**} et {**PE$_6$**, **PE$_7$**, **PE$_8$**}, en fonction de la position du candidat. Après 4 cycles d'horloges, PE1 et PE2 fournissent les deux premières valeurs SAD. Le décalage au niveau de l'activation des processeurs est conservé au niveau des sorties des processeurs comme déjà constaté dans le tableau 3.3.

Au début (cycle 0), seul ***p*** et ***h*** sont disponibles, les processeurs PE1 et PE2 sont activés. ***h***(0, j-1) et ***h***(0, j) sont disponibles respectivement pour PE1 et PE2. ***p***(0, j) est diffusé pour les deux processeurs parallèlement. Au deuxième cycle, PE1 et PE2 reçoivent la deuxième colonne pour ***p***(1, j), ***h***(1, j-1) et ***h***(1, j). Trois nouveaux processeurs sont activés (PE3, PE4, PE5).

Comme illustré en figure 3.16 et dans le tableau 3.3, la première ligne du macro-bloc courant est disponible pour ces trois processeurs. Les registres jouent un rôle essentiel dans la synchronisation des données. PE3 et PE5 calculent les distorsions des positions D, cependant le PE4 se charge de calculer la distorsion de la position V. Au troisième cycle, tous les processeurs seront activés. Chaque PE nécessite six cycles pour réaliser la tentative de mise en correspondance d'un macro-bloc de taille 4x4. Les premiers scores SAD, résultant de PE1 et PE2 sont alors disponibles au cycle 5. Au 7ème cycle, tous les processeurs fournissent leurs valeurs du SAD.

Dans le cas du processus quart de pixel, selon le meilleur point trouvé pendant le raffinement demi-pixel, 9 lignes seront envoyées, une par une et à chaque cycle afin d'obtenir l'espace interpolé requis pour le raffinement quart de pixel. Le même ordonnancement que celui du demi-pixel est appliqué.

3.3.1.3. Processeur élémentaire et unité de comparaison

L'architecture séquentielle et celle pipeline mettent respectivement en œuvre une et deux matrices de 8 processeurs élémentaires (PE). La figure 3.18 représente l'architecture interne d'un processeur élémentaire. Elle est composée de 16 unités de calcul montées en parallèles, 4 étages d'additionneurs fonctionnant en pipeline, un ensemble de 16 accumulateurs et un multiplexeur à la sortie. L'avantage de cette unité est sa modularité : elle est capable de calculer les valeurs SAD pour différentes tailles de macro-blocs. Cette unité permet également

de réaliser le traitement en parallèle de deux blocs de largeur 8 et jusqu'à quatre blocs en parallèle de largeur 4, ce qui explique les 4 sorties (S_0, S_1, S_2, S_3). Cependant ce mode de fonctionnement implique de pouvoir accéder à plusieurs sous-blocs simultanément et surtout de modifier l'unité d'interpolation. Cette spécificité permet une très nette amélioration des performances de calcul en exploitant une partie du parallélisme potentielle de données. Ce mode et les modifications architecturales nécessaires au niveau de l'interpolation seront détaillés en section 3.3.2. Cette unité est très efficace et flexible pour garantir le parallélisme introduit par la suite dans les architectures proposées pour l'estimation subpixélique.

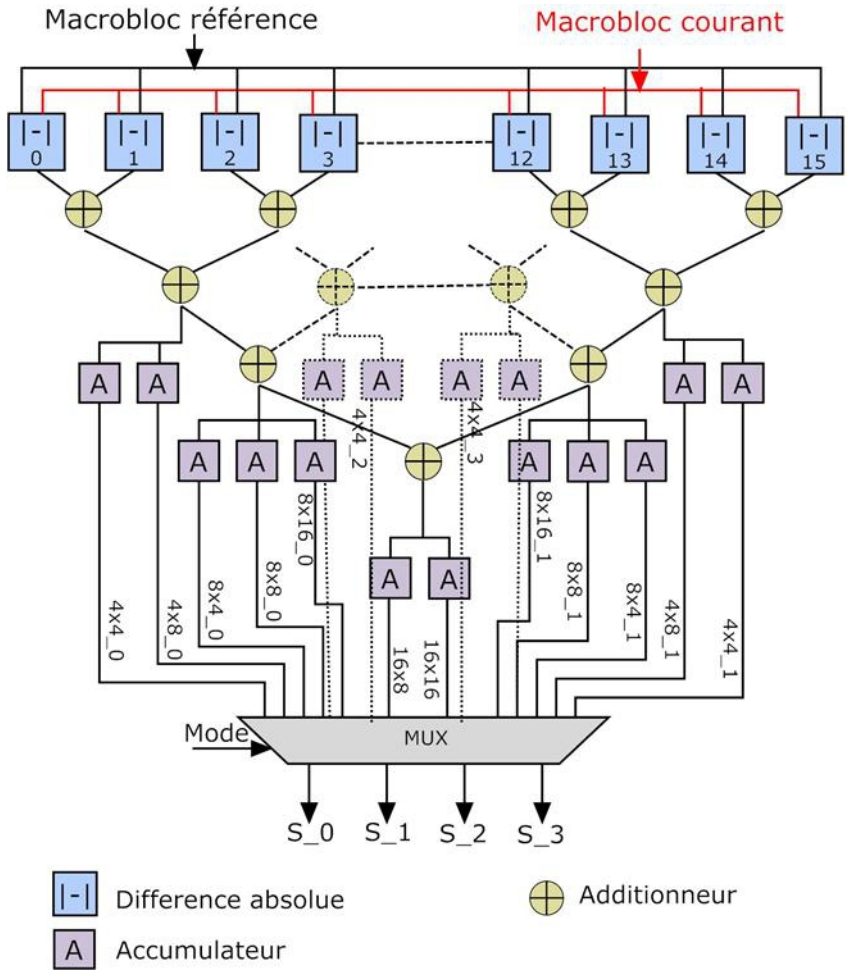


Figure 3. 18 : Architecture interne d'une unité de processeur subpixélique

Un processeur élémentaire effectue la différence en valeur absolue entre deux pixels du macro-bloc courant et celui de la fenêtre de recherche. Les additionneurs montés en parallèle et en pipeline servent à accumuler les valeurs du SAD temporaires à chaque étage et à chaque cycle. En outre, les accumulateurs accumulent les valeurs du SAD en déterminant les valeurs selon le mode choisi (4, 8 ou 16) qui correspond directement à la largeur du sous bloc traité.

Pour le mode N= (4, 8 ou 16), cette unité nécessite N cycles pour fournir les résultats. Cette architecture présente l'inconvénient, pour les modes 4 et 8, d'avoir un taux d'exploitation du matériel faible (jusqu'à 30%). Toutefois, le gain en surface de silicium en profitant de la réutilisation de logique et en temps de traitement obtenu grâce au parallélisme est remarquable.

Etant donné le parallélisme introduit dans ces architectures, le nombre d'unités de comparaison est le même que le nombre de sorties SAD fournis par l'unité des processeurs. Les résultats sont transmis à l'unité de décision qui se charge d'envoyer les vecteurs de mouvement en respectant la configuration demandée (taille de bloc variable, pixélique ou subpixélique).

3.3.2. Modification de l'unité d'interpolation : pour une architecture subpixélique parallèle

Plusieurs architectures concernant l'unité d'interpolation ont été proposées dans la littérature afin de minimiser le temps de calcul (en augmentant le parallélisme) ainsi que les besoins en termes de mémorisation des données temporaires. L'idée directrice de ces travaux [YGI06][TC11] consiste à modifier la largeur de l'unité d'interpolation afin de permettre le traitement simultané de 2 sous blocs de taille 4x4 ou 4x8. De la même façon, nous proposons un traitement simultané de plusieurs sous-blocs. Avec l'architecture proposée, il est alors ainsi possible de traiter jusqu'à 4 sous blocs de largeur 4 (4x4 et 4x8) et encore de 2 sous blocs de largeur 8 (ou de largeur inférieur) tout en se conformant aux contraintes imposées par le pipeline.

3.3.2.1. Unité d'interpolation modulaire

L'unité d'interpolation est le cœur de l'estimation subpixélique. Elle consomme plus de 40% des ressources utilisés et du temps de traitement. Par conséquent, pour améliorer la phase de raffinement subpixélique, il faut se focaliser sur l'optimisation de l'unité d'interpolation. Nous avons porté nos efforts sur l'exploitation du parallélisme et du pipeline, tout en conservant à l'esprit la nécessité d'avoir un taux élevé d'utilisation des ressources matérielles implantées.

Nous avons proposé pour cela deux versions de l'unité d'interpolation, respectant le compromis entre le temps de traitement, les calculs redondants et le taux d'utilisation du matériel. Ces deux architectures sont flexibles et compatibles avec les différentes

architectures d'estimateurs de mouvement subpixéliques. Elles sont inspirées des architectures proposées par Chen [CHC04] et Yang [YGI06]. Toutes ces architectures sont basées sur les filtres FIR-H et FIR-V déjà mentionnés.

Avant de donner les caractéristiques de nos implantations, nous allons passer en revues les éléments clefs des architectures d'interpolation existantes. Le tableau 3.4 et les figures (3.19, 3.20, 3.21) résument les caractéristiques de ces architectures.

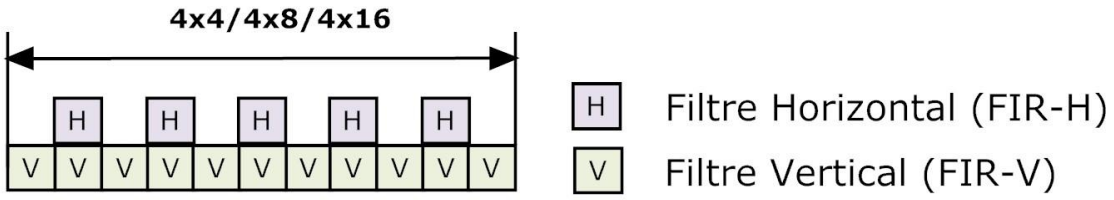


Figure 3. 19: Architecture d'unité d'interpolation de Chen

Chen [CHC04] a été l'un des premiers à proposer une architecture pour l'unité d'interpolation. Cette unité est capable de traiter un macro-bloc d'une taille maximale de 4. Pour traiter tous les sous-blocs de taille supérieure, il suffit de les décomposer en sous-blocs de taille 4 afin de les traiter séquentiellement. Comme déjà montré dans le tableau 3.4, pour interpoler un bloc de taille 4x4, Chen utilise 10 cycles. Ce nombre représente le nombre de lignes (3+4+3) minimum nécessaires pour interpoler la zone utile. En fait, un macro-bloc de taille 16x16 est composé de 16 sous-blocs de taille 4x4, ce qui explique les 160 cycles nécessaires pour interpoler les blocs de taille 4x4. De même, pour la suite des sous-blocs, la décomposition en plusieurs éléments de largeur 4 est indispensable. Par exemple, pour la taille 16x16, une zone de 22x22 centrée alentour du macro-bloc est requise. Pour calculer les coefficients demi-pixels, ce macro-bloc est divisé en 4 éléments de 4x16 pixels, où chaque élément est traité indépendamment aux autres. Un élément de taille 4x16 nécessite une zone de (3+4+3)x(3+16+3) soit 10x22 pixels, ce qui explique les 22 cycles pour achever l'interpolation. De ce fait, pour calculer les coefficients des demi-pixels pour les 41 sous-blocs, 832 cycles sont requis. Dans cette architecture, le temps de traitement est élevé, ce qui peut présenter un inconvénient majeur pour l'estimation de mouvement basée sur les algorithmes rapides

Pour réduire le temps de traitement de la phase d'estimation de mouvement subpixélique, Yang et al [YGI06] ont proposé une nouvelle architecture (décrite dans le chapitre 2) basé sur une unité d'interpolation large à 16 pixels (figure 3.20). De plus cette architecture est capable de traiter deux sous-blocs de largeur 4 simultanément, ce qui permet une réduction de 50% en nombre des cycles d'horloge, 56 et 80 au lieu de 112 et 160 respectivement pour les sous blocs 4x4 et 4x8 au niveau de ces sous-blocs. Elle permet ainsi de traiter l'ensemble des 41 sous blocs en 395 cycles. L'élargissement de l'unité de l'interpolation et la mise en œuvre de traitements parallèles pour certains sous-blocs conduisent, naturellement, à une augmentation au niveau ressources par rapport à la solution proposée par Chen.

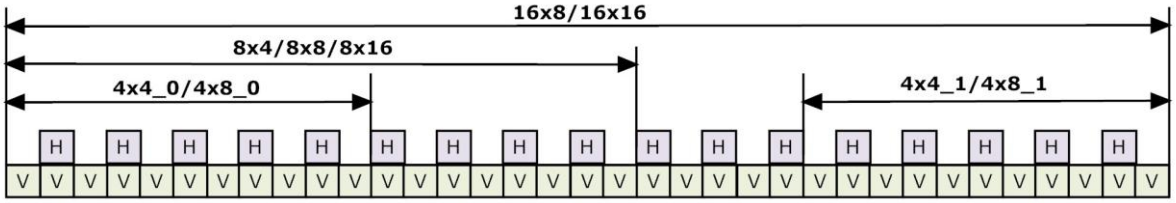


Figure 3. 20: Architecture d'unité d'interpolation de Yang

Dans le même contexte Thanga Ta et al [TC11] ont présenté une architecture basée sur une unité d'interpolation de largeur 8 (figure 3.21), pouvant traiter tous les sous blocs possibles en 552 cycles. Pour calculer les coefficients de demi-pixels pour les sous-blocs de largeur 16, une décomposition de ces derniers est nécessaire ainsi que deux phases d'estimation. Cette solution est classée intermédiaire du point de vue des performances par rapport aux deux architectures citées précédemment.

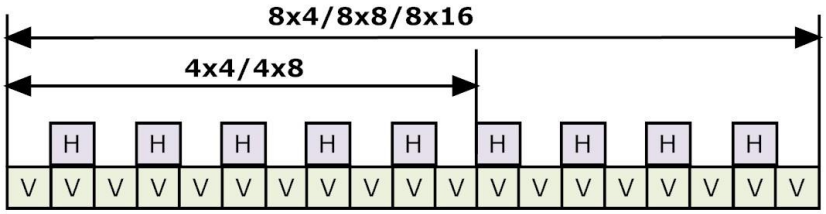
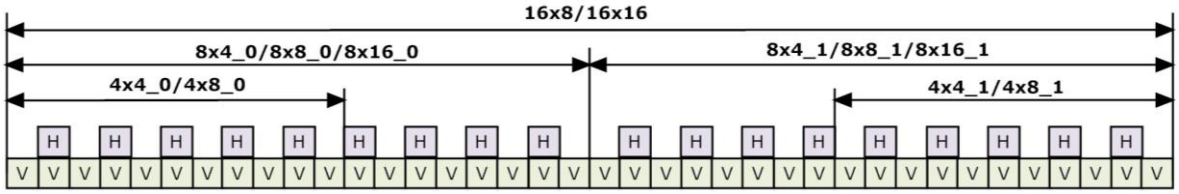
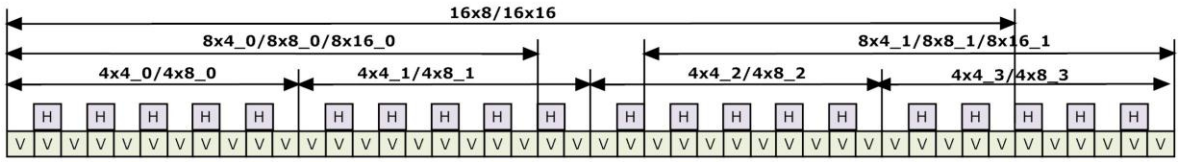


Figure 3. 21: Architecture d'unité d'interpolation de Thanga Ta

Afin d'accélérer l'étape d'interpolation, nous avons proposé deux nouvelles unités d'interpolation représentées par la figure 3.22. L'objectif principal de notre architecture était de mieux exploiter le parallélisme intrinsèque du traitement et de diminuer alors le temps de l'estimation.



a) Version 1



b) Version 2

Figure 3. 22: a) Architecture d'unité d'interpolation proposée Version 1, b) Architecture d'unité d'interpolation proposée Version 2

La première version est capable de traiter tous les sous blocs en 305 cycles soit une réduction importante du nombre de cycles, de l'ordre de 172%, 30%, 80% respectivement par rapport à Chen, Yang et Thanga Ta. Cette première version est basée sur deux unités d'interpolation de largeur 8 qui sont concaténées pour permettre de traiter deux sous-blocs (8x8, 8x16, 4x8, 4x4) en parallèle. Ceci nous permet de diviser le nombre de cycles d'horloge par deux pour tous les sous blocs de largeur inférieure ou égale à 8.

Une deuxième version plus performante que la première est proposée. Cette architecture est une concaténation de quatre unités d'interpolation de largeur 4. Dans cette architecture, le niveau de parallélisme est augmenté, et il est alors possible de traiter simultanément jusqu'à quatre sous blocs de largeur 4, ce qui permet un gain significatif en termes de cycles par rapport aux autres architectures.

Afin d'obtenir les fonctionnalités décrites ci-dessus, le nombre de filtres nécessaire à la réalisation de l'unité d'interpolation ainsi que leur association sont modifiés. Les architectures obtenues ainsi que celles faisant références, à savoir celles de Chen, Yang et de Thanga Ta, sont représentées successivement par les figures (3.19, 3.20, 3.21, 3.22).

Types sous-bloc Nombre des blocs		Chen		Yang		Thanga Ta		Notre Architecture			
								version 1		version 2	
		cycles/ bloc	total cycles	cycles/ bloc	total cycles	cycles/ bloc	total cycles	cycles/ bloc	total cycles	cycles/ bloc	total cycles
16x16	1	22x4	88	22x1	22	22x2	44	22x1	22	22	22
16x8	2	14x4	112	14x1	28	14x2	56	14x1	28	14	28
8x16	2	22x2	88	22x1	44	22x1	44	22÷2	22	22÷2	22
8x8	4	14x2	112	14x1	56	14x1	56	14÷2	28	14÷2	28
8x4	8	10x2	160	10x1	80	10x1	80	10÷2	40	10÷2	40
4x8	8	14x1	112	14÷2	56	14x1	112	14÷2	56	14÷4	28
4x4	16	10x1	160	10÷2	80	10x1	160	10÷2	80	10÷4	40
Latence		NA		29		NA		29		29	
Total	41	832		395		552		305		237	

Table 3. 4: Comparaison des architectures d'interpolation

Le tableau 3.4 récapitule les performances en termes de nombre de cycles pour différentes unités d'interpolation permettant le raffinement des 41 vecteurs de mouvements pour chaque macro-bloc. Le nombre total de cycles est la somme du nombre de cycles nécessaire pour chaque taille de bloc, auquel on ajoute le nombre de cycle de latence lorsqu'il est connu. Ainsi, pour la première version, nous obtenons un gain d'un facteur 2,72 par rapport à Chen, 1,29 par rapport à Yang, et 1,71 par rapport à Thanga Ta. Pour la deuxième version, le gain est encore plus significatif (respectivement 3,51, 1,66 et 2,32).

3.3.2.2. Calculs redondants et utilisation des ressources

Comme nous l'avons écrit, il est important d'utiliser les ressources matérielles de manière efficace et d'éviter la redondance de calculs, c'est pourquoi nous les avons évaluées et comparées à celles des autres architectures.

Comme déjà illustré par la table 3.5, chaque unité d'interpolation est décomposée en certain nombre de filtres horizontaux et verticaux. Les architectures matérielles pour ces types de filtres sont décrites dans la section précédente. Pour une unité d'interpolation de largeur N (N=4,8 ou 16), (N+1) FIR-H et (2xN+3) FIR-V sont nécessaires, soit (3xN+4) filtre FIR au total. Si on augmente la largeur de l'interpolation, une augmentation du nombre des FIR est requise pour assurer les calculs des coefficients demi-pixels. L'unité d'interpolation proposée par Chen avec N=4 est la moins coûteuse et ne consomme que 16 filtres FIR et 9 PU. Une

unité légèrement large avec N=8 est proposée par Thanga Ta, nécessitant 28 FIR et12 PU. D'autre part Yang utilise la largeur maximale (N=16) ce qui provoque une augmentation importante des ressources nécessaire : 52 FIR et 36 PU.

	Chen	Yang	Thanga Ta	Notre	
				version 1	version 2
Nombre des FIR-H	5	17	9	18	20
Nombre des FIR-V	11	35	19	38	44
Total FIR	16	52	28	56	64
Nombre des PU 4x4	9	36	12	32	32

Table 3. 5: Ressources matérielles nécessaires

De même, dans notre architecture, la concaténation de deux unités de largeur 8 dans la première version conduit à 56 FIR et 32 PU. Toutefois une augmentation de 7% du nombre de filtres FIR combinée à une diminution de 12 % de nombre des PU par rapport à Yang permet une accélération globale de 30%, ce qui prouve l'efficacité de notre architecture.

La deuxième version, la plus rapide, nécessite 64 filtres FIR et 32 PU. Cette dernière architecture est donc logiquement plus gourmande en ressources matérielles, ce qui se justifie par l'accélération qu'elle permet.

Nous avons également comparé le compromis entre le taux d'utilisation du matériel et les calculs redondants de chacune des architectures proposées.

Chen décompose chaque partition ou sous-bloc de taille NxM (N, M = 4,8 ou 16) en N/4 sous-blocs de taille 4xM. Cette décomposition permet à une unité d'interpolation de largeur 4 (figure 3.19) de traiter d'une manière séquentielle tous les sous-blocs. Comme illustré dans le tableau 3.6, l'utilisation du matériel est alors de l'ordre de 100%. Néanmoins, si la largeur de sous-bloc dépasse quatre, ceci conduit à un calcul redondant de 6x(N/4 - 1) colonnes. Dans cette configuration, plus la largeur du sous-bloc augmente plus le calcul redondant augmente. La figure 3.23 illustre les calculs redondants qui apparaissent durant le traitement du macro-bloc de taille 16x16. Les parties hachurées représentent les colonnes redondantes qui sont égales à 18.

Pour résoudre le problème des calculs redondants, Yang propose une unité large permettant de traiter tous les sous-blocs sans qu'il soit nécessaire de refaire des calculs déjà réalisés, ce qui entraîne une utilisation moyenne des ressources matérielle dans les sous-blocs de largeur 8 et 4. L'architecture proposée par Thanga Ta tente d'équilibrer la relation entre le taux d'utilisation du matériel et la minimisation des calculs redondants.

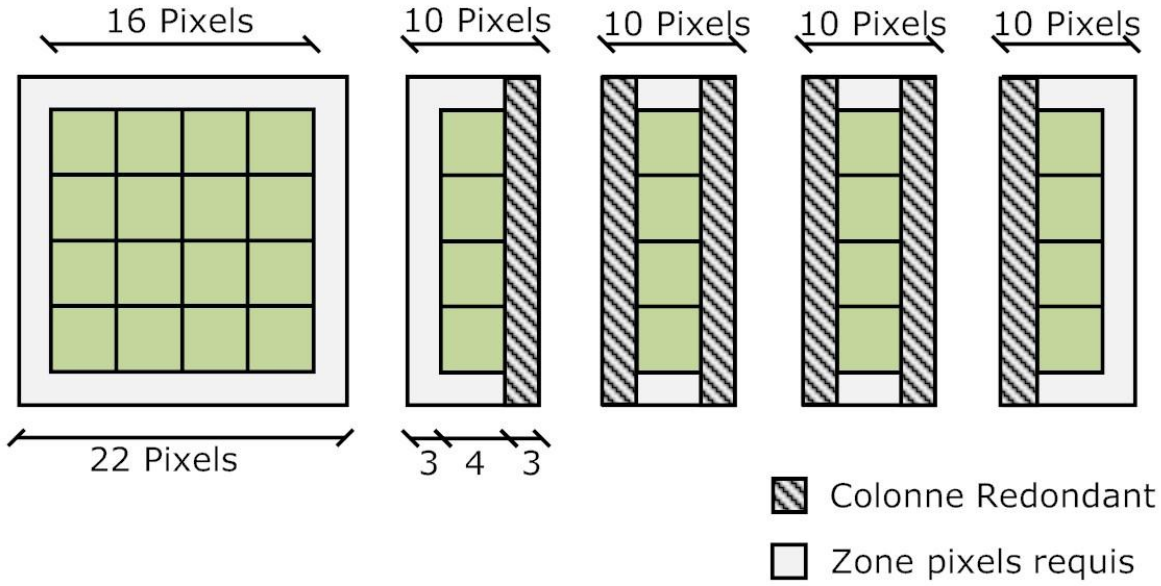


Figure 3. 23 : Décomposition du macro-bloc 16x16 en quatre 4x16 sous-blocs

Les versions que nous avons proposées évitent totalement les calculs redondants et conservent un taux d'utilisation des ressources matérielles de l'ordre de 95%. Les performances obtenues montrent bien l'efficacité de nos architectures.

			N=16	N=8	N=4
Chen		Rédondance (col)	18	6	0
		utilisé (%)	100	100	100
Yang		Rédondance (col)	0	0	0
		utilisé (%)	100	53	61
Thang Ta		Rédondance (col)	6	0	0
		utilisé (%)	100	100	57
Notre	version 1	Rédondance (col)	0	0	0
		utilisé (%)	94	100	57
	version 2	Rédondance (col)	0	0	0
		utilisé (%)	95	80	100

Table 3. 6: Taux d'utilisation des ressources et redondance des calculs

3.3.2.3. Modification de l'architecture globale

Le calcul en parallèle de différents sous blocs lors de la phase subpixélique présente malgré tout un inconvénient majeur. En effet, il est alors nécessaire d'accéder simultanément à plusieurs zones disjointes dans la mémoire. Par conséquent la mémoire destinée à recevoir la fenêtre de recherche doit être décomposée en plusieurs bancs accessibles de manière indépendante et sa taille augmentée. Dans chacun de ces bancs, une copie de la fenêtre de recherche doit alors être disponible. Bien que le concept de faisabilité concernant le traitement de 4 blocs en parallèle ait été démontré dans les paragraphes précédents, nous nous limiterons dans nos implantations à des architectures permettant le traitement simultané de deux sous blocs dans un souci d'économie des ressources mémoires et matérielles. Nous utiliserons, par conséquent, exclusivement l'unité d'interpolation notée version 1. La Figure 3.24 représente l'architecture globale ainsi obtenue.

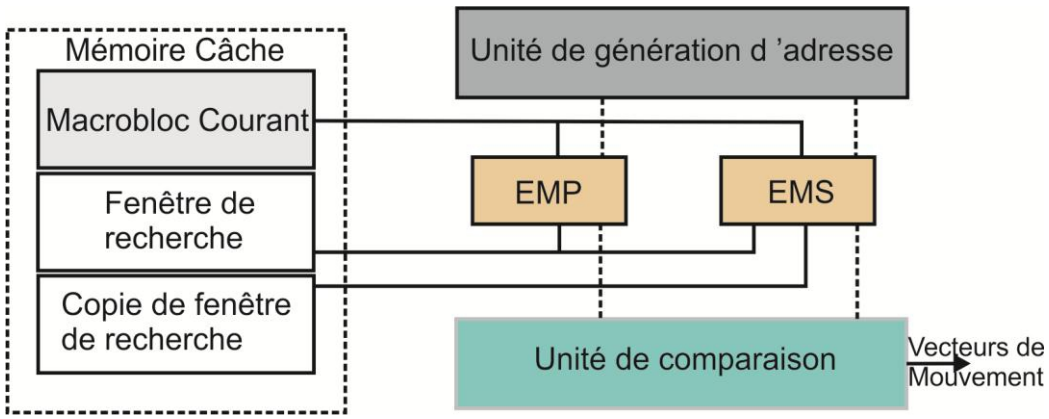


Figure 3. 24 : Schéma bloc global d'estimateur de mouvement

3.3.3. Implantations matérielles

La table 3.7 résume les résultats de synthèse concernant deux implantations d'estimateur permettant le raffinement demi-pixels et respectivement basées les deux architectures d'interpolation proposées précédemment. Une comparaison avec trois architectures à hautes performances conçues par Chen [CHC04] [CCH+06], Yang [YGI06] et Thanga Ta [TC11] est donnée table 3.8. L'estimateur de mouvement subpixélique a été implanté sur un FPGA Xilinx de type Virtex 6 (xc6vlx240t-3ff1759). En raison de ces résultats et des ressources mémoires nécessaires, seule la version 1 de l'interpolation a été utilisée pour le raffinement quart de pixel.

Estimateur de Mouvement demi-pixel (xc6vlx240t-3ff784)			
	Interpolation Version 1	Interpolation Version 2	Logique disponible
Nbre de slices	9542 (3%)	14426 (4%)	301440
Nbre de LUT	14840 (9%)	22809 (15%)	150720
Nbre Bram	16	24	46
Fréquence	310	299	

Table 3. 7: Résultats de synthèse pour l'estimateur demi-pixel

En ce qui concerne l'architecture finale permettant le raffinement subpixélique (demi et quart de pixel), nous nous sommes limités à trois implantations. La première est séquentielle et basée sur l'unité d'interpolation (version 1). La deuxième est la structure pipeline utilisant la même unité d'interpolation. Ces deux versions permettent donc le traitement simultané de deux sous blocs de largueur 4 ou 8 pixels. La troisième implantation proposée, a été développée dans un souci de faciliter la comparaison avec l'architecture référence de Yang. En effet, cette structure basée sur la même unité d'interpolation que l'architecture de référence, permet de traiter deux sous blocs de largueur 4 pixels. Cependant, contrairement à Yang, une structure pipeline a été implantée. D'autre part, les architectures de références considérées sont basées sur des cibles de type ASIC possédant une technologie plus ancienne. Il nous paraît ainsi juste de pouvoir comparer l'adaptation de cette implantation sur une cible identique aux solutions proposées.

Les résultats reportés dans le tableau 3.8 montrent que nos implantations sont très performantes en comparaison aux architectures de Chen, Yang et Thanga Ta. Par ailleurs, elles présentent un meilleur compromis temps/surface.

D'un point de vue performances de codage, les solutions architecturales proposées permettent d'atteindre les plus forts débits en termes de nombre de MacroBlocs traités par seconde à la fois pour le raffinement demi pixel mais pour celui quart de pixel. Ainsi l'architecture séquentielle permet d'atteindre 649 K MacroBlocs/s pour le raffinement demi pixel, soit un facteur d'accélération respectivement de 6,49, 1,30 et 1,25 obtenu par rapport

aux architectures de Chen, Yang et Thang Ta. En ce qui concerne l'architecture pipeline, elle permet d'atteindre 538 K MacroBlocs/s pour le raffinement quart de pixel, soit un facteur d'accélération respectivement de 10,98, 2,15 et 2,07 obtenu par rapport aux mêmes architectures.

Le nombre de cycles obtenu précédemment par l'analyse des structures doit être en effet pondéré par les fréquences de fonctionnement obtenues. Les implantations de Yang et de Ta basées sur des composants de type ASIC atteignent des fréquences supérieures à 280 MHz alors que toutes nos implantations possèdent des fréquences inférieures à 200 MHz.
La version séquentielle est plus performante en termes de fréquences de fonctionnement et utilisation des ressources. Cependant l'utilisation d'une version pipeline réduit le temps d'exécution de l'ordre de 40% par rapport à la structure séquentielle, tout en augmentant les ressources matérielle de manière raisonnable (environ 15% du composant est utilisé).

L'implantation de l'architecture basée sur l'unité d'interpolation de Yang (mais utilisant une structure pipeline) permet d'améliorer les performances de l'architecture originale pour un raffinement quart de pixel. Cependant elle ne parvient pas à égaler les performances pour raffinement demi pixel. La différence de fréquence de fonctionnement est à l'origine de la chute de performances. Nous l'imputons à la technologie utilisée.

	Chen	Yang	Thanga Ta	Architectures proposées		
				Interpolation Version 1		Interpolation type Yang
				séquentielle	pipeline	Pipeline
Tech (µm)	TSMC 0.13	TSMC 0.18	TSMC 0.18	Virtex6 0.04		Virtex6 0.04
Porte (k)	79.3	188.45	93.7	NA	NA	NA
Slice	NA	NA	NA	10676	14932	13540
LUT	NA	NA	NA	15541	20722	18034
BRAM	NA	NA	NA	36	36	36
Freq (MHz)	100	285	290	198	180	175
Cycles	1664	790	1104	610	334	417
Débit **1/2** (K MacroBloc/s)	100	500	520	649	590	443
Débit **1/4** (K MacroBloc/s)	49	250	260	325	538	419

Table 3. 8: Comparaison des implantations matérielles d'estimateur du mouvement subpixélique

Par ailleurs, la comparaison des deux versions pipeline proposées démontre un avantage significatif pour la solution permettant le traitement simultané des sous blocs de largeur égale à 8. Ainsi cette architecture permet d'augmenter, de manière significative, les performances en débit (facteur d'accélération respectivement égal à 1,33 et 1,28 pour les raffinements demi et quart de pixel) pour une augmentation moindre des ressources matérielles (< 15 %).

3.4.Conclusion

Dans ce chapitre, nous avons détaillé l'architecture de l'estimateur de mouvement configurable que nous avons implantée, et nous l'avons comparée à plusieurs architectures faisant référence dans le domaine de la compression. En ce qui concerne la phase pixélique, nous avons proposé une architecture unifiée capable de supporter différentes stratégies. En ce qui concerne l'estimation de mouvement subpixélique, nous avons focalisé notre travail sur l'étude et l'optimisation architecturale de la brique de base qu'est l'interpolation. Nous avons proposé deux versions d'interpolation efficaces et performantes capables de traiter en parallèle plusieurs sous-blocs simultanément, ce qui permet une réduction importante au niveau temps de traitement. L'amélioration des temps d'exécution confirme l'intérêt de l'optimisation de l'interpolation. L'accès aux données en utilisant les mémoires à accès parallèle est très important, un gain en performance de plus de 40% est obtenu avec une architecture pipeline, et ce gain est d'autant plus important que l'unité d'interpolation soit plus large.

La comparaison avec des travaux antérieurs a montré les avantages et parfois les limites des solutions que nous avons proposées, que ce soit en terme de vitesse, de ressources utilisées, mais surtout de flexibilité.

Chapitre IV : Estimation de mouvement : Description flot de données

4.1.Introduction

Au cours de notre travail de thèse, nous avons abordé la modélisation architecturale à l'aide d'une description manuelle en VHDL. Ce type de description a largement fait ses preuves en ce qui concerne l'efficacité des architectures obtenues, mais demande aussi un temps de développement parfois très important, et n'apporte pas un niveau d'abstraction très élevé. Les descriptions obtenues sont ainsi très proches du matériel, presque au même titre qu'un langage assembleur est proche du microprocesseur pour lequel il a été conçu. Nous avons ainsi réussi à obtenir et valider des propositions d'architecture qui présentent des avancées par rapport à l'état de l'art. Le Laboratoire Le2i s'intéresse toutefois depuis plusieurs années déjà aux recherches liées aux outils permettant de décrire les architectures à un niveau d'abstraction plus élevé, ce qui répond à un des besoins fondamentaux des développeurs, ceux-ci cherchant sans cesse à réduire le temps de développement qui devient prohibitif avec l'accroissement de la complexité et les nouveaux besoins des applications multimédia. Dans ce contexte, plusieurs langages de description de haut niveau ont été considérés. Nous pouvons citer notamment : SystemC [IEE05], CAL [EJ03], Marte [OMG09]. Ce type de modélisation associé à des outils de prototypage rapide permet de réaliser la synthèse, nommée High LevelSynthesis (HLS), de descriptions matérielles (VHDL/Verilog) ou logicielles (C/C++, Java). Ainsi, le concepteur peut repousser le moment du partitionnement matériel/logiciel du système. On peut encore noter que plusieurs thèses [BOU05] [FAR08] ont par exemple déjà été consacrées au Le2i à des évaluations liées à un outil comme Syndex[HYP+97], développé à l'INRIA [SYN]. Bien d'autres outils existent, comme GAUT [GAU], qui génère du VHDL à partir d'une description C/C++. Toutefois, comme nous l'avons précisé dans l'introduction générale de ce document, notre travail de thèse a été réalisé pour le projet **smart camera**. Dans ce cadre, c'est le langage CAL qui nous a été imposé, ceci étant lié à la volonté du Le2i d'évaluer les apports potentiels de cet outil dans notre contexte. Cette évaluation a donc été menée uniquement en comparaison à la description VHDL manuelle.

Dans ce chapitre nous montrons donc l'intérêt de l'utilisation du langage CAL par rapport aux langages de plus bas niveau pour le prototypage des applications, intérêt qui réside notamment dans le fait que les outils associés peuvent générer automatiquement un large panel d'interfaces standard. Nous proposerons de décrire la partie estimation de mouvement

pixélique à l'aide du langage CAL tout en profitant du parallélisme offert par ce langage, afin de générer automatiquement le code VHDL associé, ce qui permet de réduire le temps de mise sur le marché « Time to Market » (TTM). Une comparaison avec une version décrite manuellement est effectuée. Les résultats obtenus sont encourageants, montrant l'efficacité et la fiabilité de ce langage. Cependant, nous verrons que des améliorations au niveau du synthétiseur permettraient d'obtenir de meilleures performances.

4.2.Langage flot de données CAL

De nos jours, l'utilisation des données de type multimédia a énormément évolué. La progression de la communication numérique (Télévision Haute Définition) et l'arrivée de nouveaux terminaux multimédia (Smartphone, Tablette, PDA) ont étendu les applications de la représentation numérique de contenu multimédia. En 2005, Le comité MPEG (Motion Picture Expert Group) a donné naissance à un nouveau standard pour le codage vidéo « Reconfigurable Video Coding » (RVC). Un des objectifs principaux de MPEG/RVC est de pouvoir concevoir un décodeur à partir d'une spécification de haut niveau d'abstraction par rapport à celle des standards MPEG décrites en langage C. La flexibilité de la description à haut niveau et la rapidité de développement, sont les principales caractéristiques qui définissent ce standard.

La norme MPEG-B définit les langages qui sont utilisés dans le contexte MPEG/RVC. Un langage de description flot des données est proposé par MPEG/RVC pour assurer ce type de description.

4.2.1. Principe de base

D'une manière générale, la modélisation haut niveau de type CAL décrit l'évolution du flot de données à travers différentes fonctionnalités du système. Après le raffinement de la modélisation flot de données, la conversion de code en une description permettant une implantation matérielle (VHDL) ou logicielle (C, JAVA) est respectivement rendue possible par les outils OpenDf et Orcc. Le flot de développement est illustré figure 4.1. Dans ce travail, nous avons évalué l'impact de cette méthodologie sur la phase de développement de l'estimateur de mouvement au niveau pixélique. Ainsi l'architecture en charge de ce traitement a été décrite en langage flot de données RVC/CAL, et à partir de ce point d'entrée et des outils de prototypage rapide associés [RWR+08][JMP+08a],une modélisation

équivalente a été obtenue et évaluée. Une description en code VHDL obtenue par génération automatique est ainsi comparée à notre implantation matérielle proposée dans le chapitre précédent. De plus, nous proposons une discussion concernant l'intérêt d'une implantation hétérogène matérielle/logicielle par rapport à la problématique proposée d'adaptation de la stratégie de recherche de l'estimateur de mouvement.

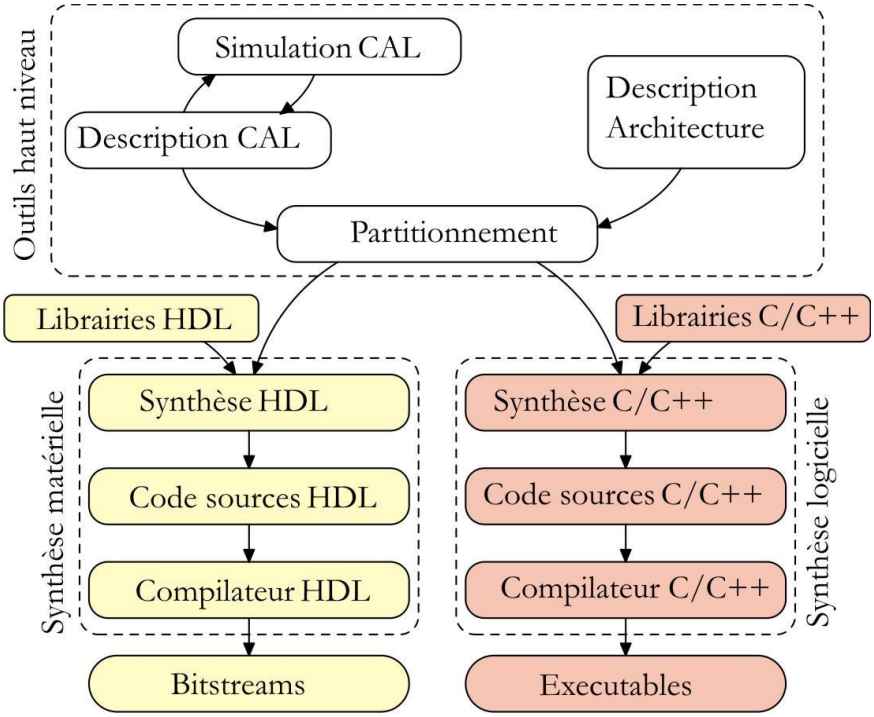


Figure 4. 1 : flot de conception

Tout d'abord, le programme en CAL est simulé et vérifié d'une façon comportementale. Ensuite, la deuxième étape sert au pré-partitionnement. Cette étape définit l'architecture support et la partition logicielle et matérielle. Les interfaces de communication et les pilotes associés peuvent être alors générés[TMD+11].

Dans ce contexte, plusieurs travaux basés sur une description flot de données ont été développés et présentés. Un décodeur vidéo MPEG-4 Simple Profile [AMH+11] [JMP+11] a été décrit en utilisant CAL : le code VHDL associé y est généré à l'aide de l'outil CAL2HDL [LMW+08b] afin de l'implémenter à base d'un FPGA. Ainsi non seulement le temps de développement est divisé par quatre par rapport à un code décrit manuellement, mais l'implantation peut se révéler, dans certains cas, plus efficace en terme de débit de sortie ou permettre une diminution des ressources matérielles.

Dans le cadre de la conception de l'estimateur du mouvement avec précision pixélique, le gain en temps de développement permis par l'utilisation de la méthodologie basé sur RVC-CAL est clairement mis en évidence dans [DTM+09][EDA+11].

4.2.2. Le formalisme CAL-RVC

Ce langage est basé sur les unités fonctionnelles atomiques nommées « Acteurs ». Cette entité est autonome et peut comporter une partie opérative et un séquenceur (de type machine à états). Chaque acteur peut contenir plusieurs actions pouvant fonctionner en parallèle ou être cadencées par un séquenceur. L'approche est hiérarchique : des réseaux d'acteurs peuvent être ainsi définis.

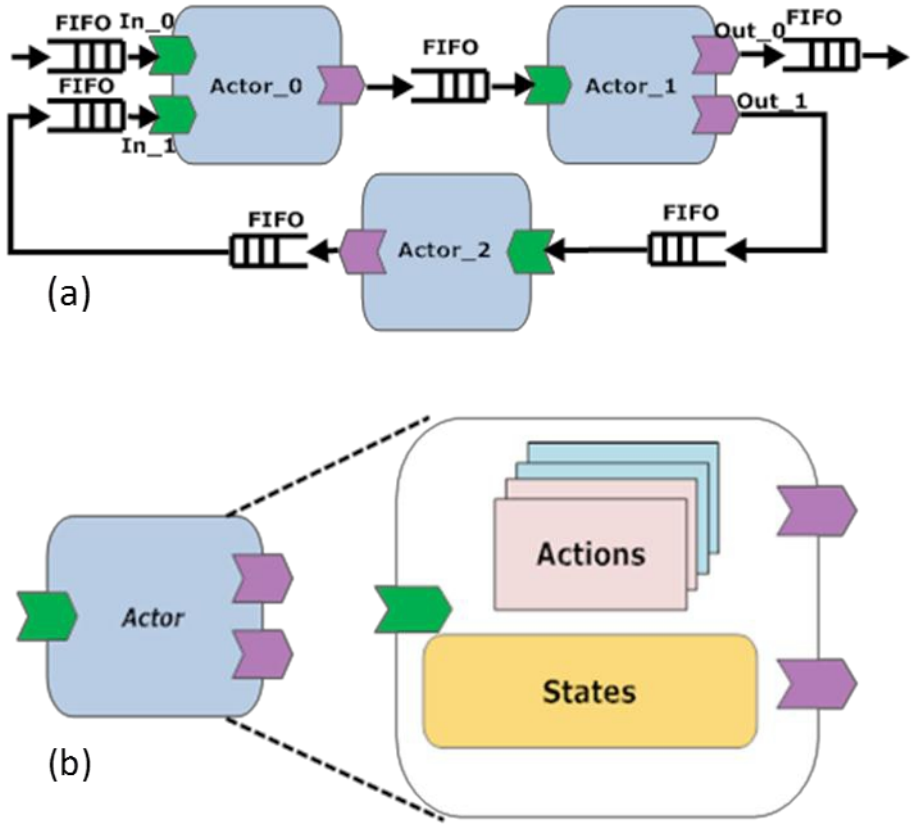


Figure 4. 2 : Modèle flot de données et structure de base d'un acteur

Le formalisme de type flot de données utilisé respecte la sémantique des « DPNs » (Data Process Networks) eux-mêmes reliés aux « KPNs » (Kahn Process Networks) [MW10]. Les liaisons entre acteurs sont unidirectionnelles et se comportent comme des files d'attente de taille infinie et de type First In First Out (FIFO). L'écriture dans une de ces liaisons est non

bloquante. L'exécution d'un acteur n'est possible que lorsque les données d'entrée sont disponibles. Comme illustre la figure 4.2, un acteur possède des ports d'entrées (In_0, In_1, etc.) et des ports de sorties (Out_0, Out_1, etc.). Ces ports sont les canaux de transfert des jetons (données) entre les différents acteurs.

Une action peut être une simple opération telle qu'une addition entre deux entrées, ou bien plus complexe comme l'illustre l'algorithme 4.1 qui décrit, à titre d'exemple, une transformation des couleurs de l'espace YUV vers l'espace RGB. L'acteur de transformation en couleur YUV2RGB possède trois ports en entrées et trois ports de sorties qui représentent respectivement les trois composantes de luminance Y et de chrominances (U et V) ainsi que les trois composantes couleurs (R, G et B). Une seule action se charge de lire les jetons présents sur les ports d'entrées qui seront utilisés par la suite dans les équations de transformation situées dans le corps de l'action, afin de fournir les résultats sur les trois ports de sorties.

L'acteur est toujours en état de fonctionnement tant que des jetons d'entrée sont disponibles. Il produit des résultats qui doivent être consommés par un acteur en aval de celui YUV2RGB.

Algorithme 4.1

```
actor yuv2rgb()
  int(size=9) y,
  int(size=9) u,
  int(size=9) v
  ==>
  int(size=8) r,
  int(size=8) g,
  int(size=8) b:
  action y:[yc], u:[uc], v:[vc] ==> r:[rc], g:[gc], b:[bc]
     var
           int(size=8) rc, int(size=8) gc, int(size=8) bc,
           double rd, double gd, double bd,
           double k1 = 1.402, double k2 = 0.3441,
           double k3 = 0.7141, double k4 = 1.772
     do
           rd:= (yc-16) + k1*(vc-128);
           gd:= (yc-16) - k2*(uc-128) - k3*(vc-128);
           bd:= (yc-16) + k4*(uc-128);

           rc:=int(rd);
           gc :=int(gd);
           bc :=int(bd);
  end
end
```

Les actions peuvent s'exécuter séquentiellement aussi bien que parallèlement (simultanément), ce qui donne la possibilité d'appliquer le parallélisme sur les données, les tâches et/ou les pipelines afin d'optimiser les performances de l'application. Cependant, l'exécution des instructions à l'intérieur d'une action est séquentielle. Comme mentionné précédemment, les acteurs peuvent comporter plusieurs actions ce qui peut provoquer des problèmes de non-détermination. Un acteur est dit non-déterministe si un port de sortie est affecté en même temps par plusieurs actions qui s'exécutent simultanément. Cela provoque un disfonctionnement global de l'acteur. Plusieurs techniques et syntaxes ont été présentées afin de résoudre les problèmes de non-détermination et de faciliter le séquencement des tâches. Parmi ces techniques, on trouve l'utilisation de « Guard » qui permet de contrôler l'activation et la désactivation d'une action. Cette technique peut résoudre le problème de non-détermination cité précédemment en ajoutant une condition nécessaire et valide pour que l'action produise sa sortie. D'autres techniques peuvent répondre à ce problème et gérer le séquencement des actions, comme les machines d'états finis (FSM) et les priorités.
Dans la section suivante, une description détaillée d'un estimateur de mouvement en CAL est présentée.

4.3.Description flot de données : Estimateur de Mouvement

Dans le contexte RVC, les performances de codage en termes de qualité et temps de traitement, doivent être adaptable aux besoins de l'utilisateur ou de l'application. Comme déjà mentionné précédemment, l'estimation de mouvement est une opération clé dans la norme H.264. Par conséquent, nous proposons de décrire en CAL-RVC un estimateur flexible permettant de choisir l'algorithme (la stratégie de recherche) selon les besoins de l'application. La description à haut niveau d'abstraction devra ainsi permettre de modifier l'algorithme de recherche de manière plus simple et plus rapide. Pour autant, nous désirons également estimer les performances intrinsèques de l'outil de conversion de code OpenDf. Cette modélisation est utilisée pour la génération automatique d'un code VHDL équivalent qui sera comparé à la description présentée dans le chapitre 3.

L'architecture proposée supporte la stratégie de recherche suivant une grille en diamant (DS) et la recherche exhaustive (FS). Le parallélisme de données déjà utilisé dans la description manuelle est repris pour la description flot de données. Ainsi, comme représenté Figure 4.3, la description flot de données utilisera une unité pour les mémoires caches, un

acteur pour l'extraction des données utiles pour l'opération comparaison lignes par lignes, un acteur de traitement pour l'opération de type SAD et les comparaisons requises, et finalement un acteur en charge de la génération de toutes les adresses requises pour chaque phase de recherche.

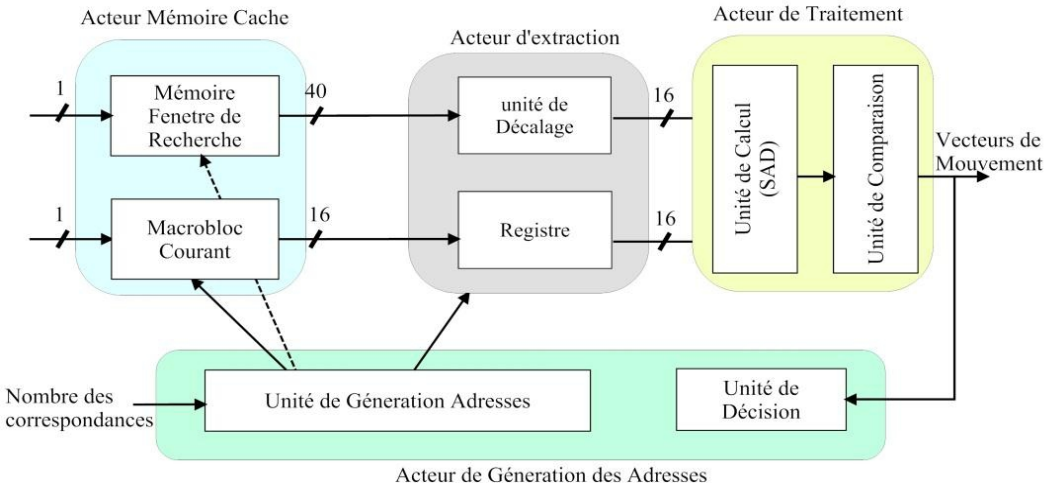


Figure 4. 3 : Le réseau d'acteurs de l'estimateur de mouvement pixélique

L'acteur traitement pourrait être lui-même décrit comme un réseau d'acteurs afin d'expliciter le parallélisme du pipeline décrit dans le chapitre 3. Cependant nous avons préféré conserver un nombre réduit d'acteurs dans cette modélisation. En effet, la multiplication des acteurs aura pour conséquence l'augmentation des ressources matérielles en particulier les mémoires FIFO nécessaires à la communication entre les différents acteurs. Nous proposons la mise en place d'un acteur unique comportant six actions (*Start*, *Read P/SW*, *Fire1*, *Fire2*, *Fire3* et *Fire4*) gérées par niveau de priorité. La première action *Start* est prioritaire sur toutes les autres actions. Cette action reçoit, sous la forme de jetons d'entrée, le nombre de positions à estimer et les adresses associées. Une nouvelle phase de recherche est prête pour déclencher le calcul des valeurs SAD associées aux adresses requises. Cette action autorise alors la réception des données et le traitement de ces dernières, tâches respectivement associées aux actions *Read P/SW* et *Fire1*. Cette autorisation est réalisée par activation de la condition dite de « guard » comme le décrit l'exemple ci-dessous :

Read_P/SW: action pa :[pat], swa :[swat] ==>

guardread_c=true, tok=true

do

pipeline_set := true;

end

```
Fire1:  action ==>
        guard (pipeline_set = true)
        do

        pipeline_set := false;
        end
```

Dans cet exemple, la réception des données pa, swa déclenche la réalisation de l'action si la condition de guard est vérifiée (ici il faut que read_c et tok soient « vrai »). L'action *Read P/SW* va quant à elle déclencher l'action *Fire1* grâce à l'activation de la condition pipeline_set. Les actions *Read P/SW* et *Fire1* sont ainsi liées, la seconde étant déclenchée au « moment » désiré par la première.

L'action Fire1 comporte la description d'un pipeline. Nous proposons un exemple de description d'une structure pipeline au sein d'une action. Deux étages sont décrits dans cet exemple, l'addition des différences diff1 et diff2 étant réalisées dans la même phase que le calcul des deux valeurs absolues stockées dans diff1 et diff2.

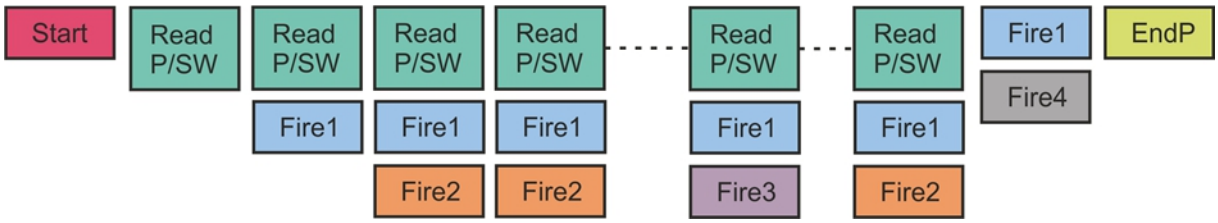
```
add11 := diff1 + diff2;

if (pp1 > pw1) then diff1:= (pp1 - pw1); else diff1:= (pw1 - pp1); end

if (pp2 > pw2) then diff2:= (pp2 - pw2); else diff2:= (pw2 - pp2); end
```

Il faut absolument noter que nous avons choisi de décrire cette structure pipeline au sein d'une action pour éviter l'utilisation d'un réseau d'acteur. Nous avons donc développé cette alternative pour optimiser ce cas particulier.

Les actions *Fire2*, *Fire3* ou *Fire4* permettent d'émettre les jetons de sorties et gérer la fin de traitement. L'ensemble des étages de la structure pipeline doit pouvoir achever le traitement de la dernière estimation. Le traitement de 16 lignes est nécessaire pour obtenir le résultat d'une tentative de mise en correspondance de blocs. Comme le montre la figure 4.4, certaines actions vont pouvoir se dérouler en parallèle pour accélérer le traitement. La description en CAL de cet acteur est en annexe.



Start : débutduprocessus de recherche,
Read P/SW : Lecture de deux lignes de 16 pixels du macrobloc courant et de la fenêtre de recherche,
Fire1 : Alimentation de l'unité des processeurs par les nouveaux pixels,
Fire2 : Accumulation des valeurs SAD,
Fire3 : Renvoi du SAD associée de l'adresse du macrobloc,
Fire4 : Vidage des étages pipelines de l'unité de traitement durant les 5 derniers cycles,
EndP : Fin duprocessus de recherche et renvoi du résultat.

Figure 4. 4: Ordonnancement des actions d'acteur SAD

4.4. Simulation et validation fonctionnelle

La validation fonctionnelle de notre modèle CAL est réalisée à l'aide d'une phase de simulation. Aux quatre acteurs assurant l'estimation de mouvement, deux acteurs (Source et Puit) sont ajoutés pour la simulation du modèle (Figure 4.5).

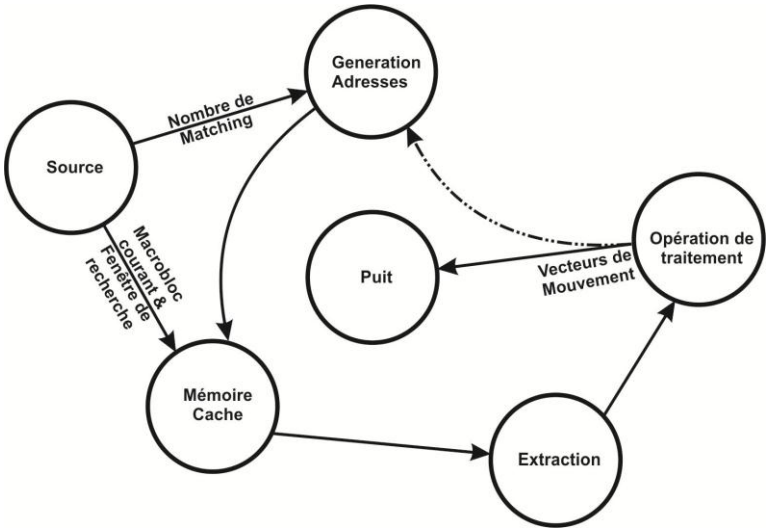


Figure 4. 5 : Modélisation sous la forme d'un graphe de flot des données d'estimateur de mouvement

L'acteur Source extrait les différents macro-blocs et zones de recherche associées des séquences d'images de la vidéo courante. L'acteur Puit permet l'affichage des vecteurs de mouvement.

Le plugin de la plateforme Orcc, nommé Graphitti, facilite l'interconnexion des réseaux d'acteurs afin d'avoir une simulation plus simple. Un exemple de cette interface est représenté figure 4.6.

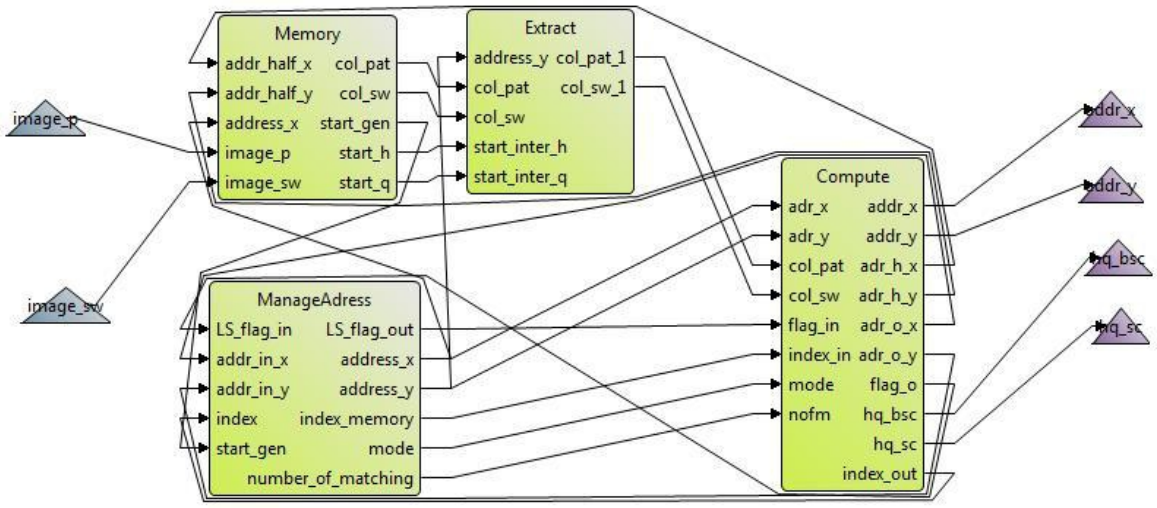


Figure 4. 6 : Interconnexion des acteurs avec Graphitti sous Orcc

4.5. Génération automatique des codes et implantation

Nous avons utilisé l'outil de génération CAL2HDL [LMW+08][JMP+08a] développé par Xilinx pour générer le code VHDL associé au modèle décrit en CAL-RVC. Après avoir réalisé la simulation fonctionnelle, nous avons procédé à la synthèse et au placement routage du code généré. L'outil utilisé est le synthétiseur XST de Xilinx ISE. La cible matérielle est un FPGA de type Virtex 6 (XC6VL25). Cette génération automatique met en évidence un certain nombre d'avantage. D'une part, outre une lisibilité subjective due intrinsèquement à la modélisation flot de données mais aussi à sa représentation graphique, le code flot de donnée est plus compact que la description VDHL manuelle. La taille du code CAL-RVC est environ trois fois plus petite que celle du code en VHDL. Le gain obtenu en termes en temps de développement et de simulation est environ quatre. D'autre part les performances de l'implantation ainsi obtenue sont encourageantes bien que légèrement dégradées par rapport à la description manuelle. Les résultats de synthèse de l'architecture sont présentés dans la

table 4.1. Ces résultats présentent les ressources matérielles nécessaires (nombre de slices, de flip flops, des LUT, et des blocs DSP utilisés) à l'implantation de l'estimateur de mouvement pixélique.

Ressources	**Utilisées**	**Disponibles**	**Occupation**
Nombre de Slices	2974	301440	<1%
Nombre de LUTs	6128	150720	4%
Nombre de BRAMs	2	416	<1%
Fréquence Maximale		195 MHz	

Table 4. 1 : Résultats de la synthèse de l'estimateur de mouvement

Acteur	**Ressources**	**Utilisées**	**Disponibles**	**Occupation**
Traitement (SAD+CMP)	*Nombre de Slices*	622	301440	<1%
	Nombre de LUTs	1128	150720	<1%
	Nombre de BRAMs	0	416	
	Fréquence Maximale		**331** MHz	
Extraction	*Nombre de Slices*	318	301440	<1%
	Nombre de LUTs	673	150720	<1%
	Nombre de BRAMs	0	416	
	Fréquence Maximale		**195** MHz	
Mémoire cache	*Nombre de Slices*	1329	301440	<1%
	Nombre de LUTs	1271	150720	<1%
	Nombre de BRAMs	2	416	
	Fréquence Maximale		**330** MHz	
Génération d'adresse	*Nombre de Slices*	757	301440	<1%
	Nombre de LUTs	775	150720	<1%
	Nombre de BRAMs	0	416	
	Fréquence Maximale		**270** MHz	

Table 4. 2 : Résultats des synthèses de chaque Acteur

Globalement le surcoût matériel apparaît relativement important puisque le nombre de blocs logiques (ou Slices) et de LUTs sont supérieurs respectivement d'un facteur 3 et 6 comparé à la description manuelle. Les débits sont quant à eux réduits d'un facteur 2,25. En

effet, les résultats de la synthèse montrent que le système fonctionne avec une fréquence maximale de 195 MHz. La diminution significative de cette fréquence par rapport aux résultats d'implantation présentés dans le chapitre 3 (fréquence maximale de 438 MHz), nous a conduit à analyser l'implantation de chaque acteur de traitement de manière séparée. Ces résultats sont présentés dans le tableau 4.2.

Les fréquences de fonctionnement de chacun des blocs sont estimées suite à la phase de placement et routage et sont ensuite confirmées par une simulation (avec une prise compte des temps de propagation) réalisée avec l'outil ModelSim.

L'analyse de ces résultats montre que ces différents acteurs ont une fréquence de fonctionnement supérieure à 270 MHz sauf celle de l'acteur d'extraction, qui estde 195 MHz. Ce module est donc responsable de la limitation globale de la structure pipeline. Comme nous le décrivons dans [DTM+09][EDA+11], nous imputons cette limitation à la traduction CAL vers HDL de certains opérateurs par l'outil OpenDf. En effet, les opérateurs de concaténation bit à bit mais aussi d'extraction d'un bit ou d'un groupe de bits n'existent pas en CAL-RVC. Il est ainsi nécessaire d'utiliser des masques et des opérateurs de décalages. Or les bus de données de notre application sont souvent de très grande largeur (sélection de 16 pixels parmi 40). L'implantation obtenue est ainsi fortement consommatrice en tables LUT parfois cascadées, ce qui a pour conséquence de faire chuter la fréquence maximum de fonctionnement. L'intégration des opérateurs manquants permettrait de résoudre au moins partiellement le problème et d'augmenter de la fréquence de fonctionnement de ce module et donc du système global.

A titre d'exemple nous donnons les opérateurs mis en jeu pour réaliser l'extraction des pixels pp1 et pp2 (sur 8 bits) d'un bus « pat » de 16 pixels (sur 16 x 8 bits soit 128 bits).

pp1 := bitand(pat,255);	pp1 <= pat (7 downto 0);
pp2 := bitand(rshift(pat,8),255);	pp2 <= pat(15 downto 0) ;
Code en CAL-RVC	Code équivalent en VHDL

L'utilisation du parallélisme de données au niveau de l'acteur de traitement comme présenté précédemment a permis d'obtenir de bonnes performances au niveau de l'implantation. La fréquence de fonctionnement de cet acteur est en effet supérieure à 330 MHz.

4.6.Conclusion

Bien que l'implantation matérielle obtenue par partir du code automatiquement généré à partir de la description flot de données ne permette d'obtenir que des performances relativement modestes par rapport à l'implantation réalisée à partir du code VHDL décrit manuellement, cette solution nous apparaît tout de même très intéressante en raison de sa flexibilité, de la compacité de la description, de la réduction du temps de développement mais aussi et surtout grâce à la possibilité de proposer une architecture hétérogène à partir d'un modèle unifié.

En effet la possibilité de générer du code en C, à partir de la même description en CAL-RVC, permet de proposer une flexibilité supplémentaire vis-à-vis de la problématique traitée. En fonction de la complexité et de la régularité de l'algorithme de recherche choisi, il est très simple de modifier la partition matérielle/logicielle et de traduire cet acteur en code C ou VHDL.

Ainsi, en ce qui concerne notre problématique particulière de compression vidéo, plusieurs solutions d'implantation peuvent être proposées très rapidement grâce aux outils de génération automatique de codes. L'approche de conception CAL facilite les tâches de validation de cette nouvelle architecture, ce qui nous permet de choisir, pour l'acteur génération des adresses (partie commande), une implantation matérielle (FPGA) ou logicielle (PC) comme illustré dans les figures 4.7 et 4.8.

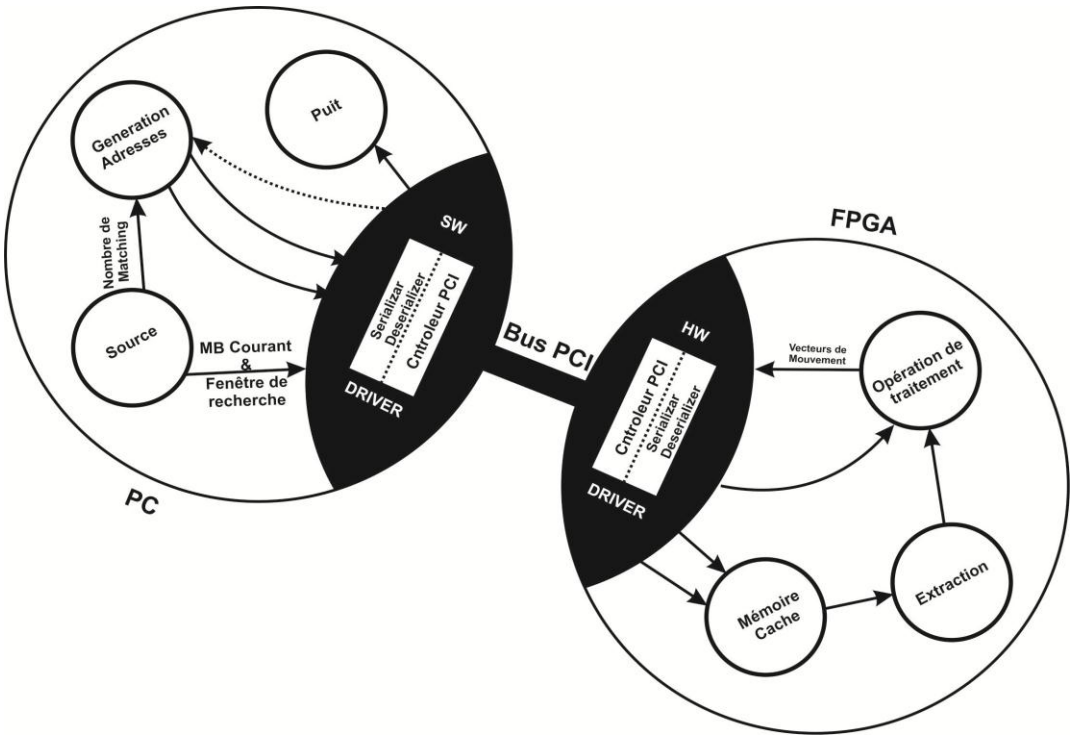


Figure 4. 7 : Implantation matérielle/logicielle de l'estimation de mouvement

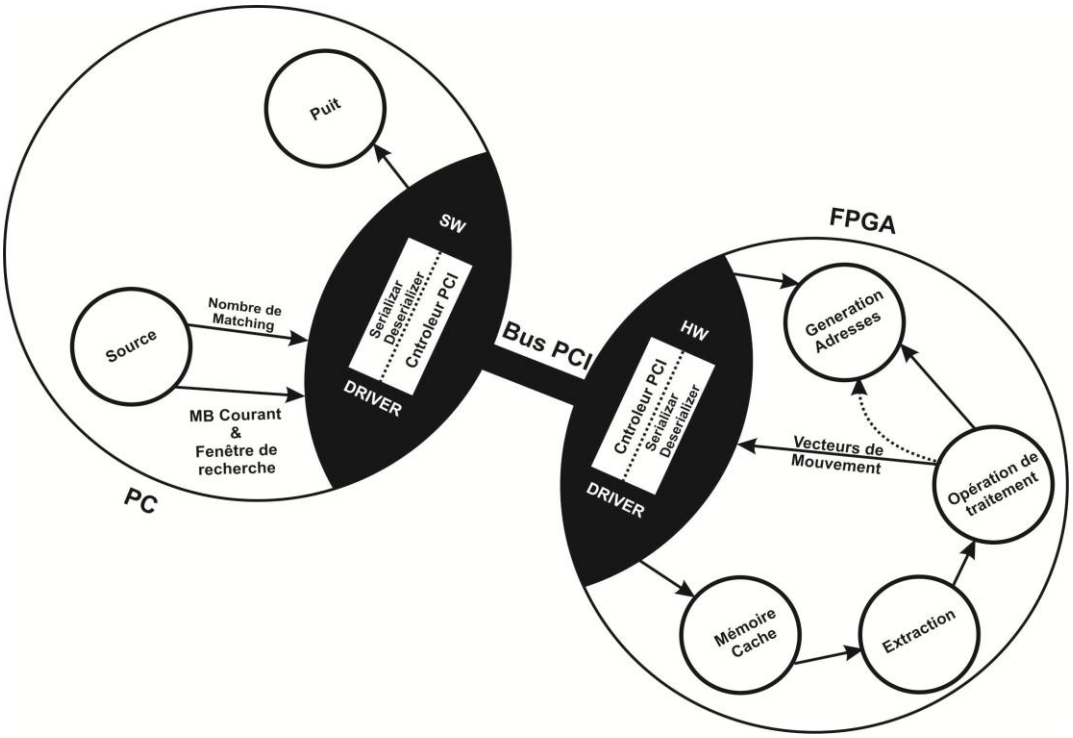


Figure 4. 8 : Implantation matérielle de l'estimation de mouvement

Les outils de génération automatique associés au formalisme CAL peuvent fournir l'architecture complète. En effet, outre les partitions matérielles et logicielles, les interfaces de communication (par exemple une interface PCI), peuvent être générées automatiquement [TMD+11][TMD+09] ce qui facilite encore l'implantation d'une architecture hétérogène, qui reste un problème traditionnellement délicat à gérer manuellement.

Conclusion et perspectives

L'objectif de cette thèse était de définir et d'implanter une architecture matérielle d'un estimateur de mouvement configurable capable de supporter différentes stratégies de recherche avec la précision désirée pour la compression vidéo adaptative. Ce besoin d'adaptabilité avait plusieurs origines. D'une part les systèmes actuels visent à s'adapter à la diversité et l'hétérogénéité des médias et des terminaux actuels. D'autre part, l'exploitation de l'information contenue dans une scène vidéo dépend de l'application visée et des besoins des utilisateurs. Cet objectif s'inscrivant de façon modeste dans le défi offert par l'évolution de la vidéo numérique qui nécessite un traitement plus rapide ainsi qu'un ratio de compression de plus en plus élevé.

La première partie de notre travail a consisté à étudier l'implantation d'estimateurs de mouvement pour la compression vidéo. Nous avons tout d'abord réalisé une étude préliminaire approfondie à propos les différents algorithmes de recherche ainsi que les architectures matérielles associées pour extraire d'une part les points de ressemblance algorithmique et dégager les blocs de traitements communs entre eux, et d'autre part de définir et proposer une architecture matérielle optimale capable de supporter différentes stratégies de recherche. Ensuite, une interprétation architecturale montre que d'une manière générale l'utilisation des architectures systoliques pour les estimateurs de mouvement offre une accélération de traitement importante mais ne favorise pas la possibilité de supporter différentes stratégies vu que ce type d'architectures est basé sur une matrice de processeurs et de registres câblés auparavant en Inter-Candidat ou Intra-Candidat permettant les calculs des SAD d'un ensemble de macro-blocs définis. Nos analyses et interprétations nous ont menés à baser notre architecture sur le traitement séquentiel de l'ensemble des points définis à chaque itération de n'importe quel algorithme. Cependant, une architecture parallèle est exigée afin

d'accélérer ce type de traitement. L'organisation de la mémoire proposée fait partie d'une amélioration au niveau de l'optimisation de taux d'accès mémoire en tenant compte de la nécessité de réduire les besoins en mémoire. Dans cette architecture nous avons utilisé la technique des mémoires dupliquées afin de sortir une ligne de la fenêtre de recherche et celle du macro-bloc courant à chaque cycle. Cette organisation été renforcée par une unité d'extraction qui assure la sélection de 16 pixels utiles de la ligne envoyée par les bancs mémoire de la fenêtre de recherche. Ceci permet à chaque cycle, d'alimenter l'unité des processeurs avec deux lignes ou colonnes de 16 pixels du macro-bloc courant et de la fenêtre de recherche. Cette architecture parallèle peut évaluer une correspondance en 16 cycles. Une nouvelle architecture d'unité de processeur à taille de bloc variable basée sur 16 processeurs élémentaires qui traitent deux lignes de 16 pixels de la fenêtre de recherche et du macro-bloc courant à chaque cycle a été proposée. Le changement d'un tel algorithme de recherche est restreint à la modification du module de génération d'adresse et exploite la totalité des blocs constituant l'estimateur de mouvement.

La deuxième partie est consacrée au raffinement subpixélique de 41 vecteurs de mouvements obtenus lors de la phase pixélique. Notre contribution porte sur la proposition et l'implantation matérielle de deux architectures (séquentielle et pipeline) efficaces et performantes capables de traiter en parallèle plusieurs sous-blocs simultanément.
L'architecture séquentielle profite de l'utilisation commune de l'unité de traitement et l'unité de comparaison pour le raffinement demi et quart de pixel afin de réduire les ressources matérielles utilisées.

L'architecture pipeline est rapide et profite du pipeline des tâches et des données afin d'accélérer la phase subpixélique, ce qui permet une réduction importante au niveau temps de traitement de l'ordre de 40% par rapport aux travaux antérieurs. Notons que l'unité d'interpolation influe sur le temps de traitement vu qu'elle consomme plus que 40 % de ce temps.

Les deux nouvelles architectures proposées visent à accélérer le temps nécessaire pour le raffinement subpixélique. La première version pourrait traiter simultanément 2 sous blocs de largeur 4 ou 8, en plus la deuxième version favorise le traitement de 4 sous blocs de largeurs de 4 et 2 sous blocs de largeur 8. Des comparaisons au niveau temps de traitement, ressources

utilisées et taux d'utilisation du matérielle avec les unités d'interpolation proposées dans les travaux antérieurs à hautes performances, montrent bien la pertinence de nos deux architectures. L'élargissement des unités d'interpolation conduit à une grande complexité matérielle au niveau du traitement et du stockage ainsi que l'extraction des données utiles d'une part, et d'autre part la simulation fonctionnelle devient de plus en plus délicate ce qui provoque un ralentissement de la durée de développement et par suite une augmentation du temps de mise sur le marché.

Dans la dernière partie, nous nous sommes intéressés à suivre une des nouvelles approches de description, à un niveau d'abstraction plus élevé que le niveau transfert de registres, ce qui permet entre autre l'accélération de temps de développement et d'améliorer la modularité de la description et donc son évolution éventuelle. Une description en langage CAL de l'estimateur de mouvement pixélique est présentée afin de générer automatiquement le code associé en VHDL ou en C. Le temps de développement est divisé par 4 par rapport à la version manuelle en VHDL. Toutefois, les résultats de synthèse obtenus avec le code manuel dominent ceux obtenus avec la génération automatique. Cependant, au niveau temps et flexibilité de développement, la description en CAL reste plus efficace, et l'on peut supposer que ces outils évolueront rapidement et permettront bientôt d'atteindre des performances équivalentes, si ce n'est supérieures, aux descriptions manuelles.

Au moment de clore ce manuscrit, des perspectives prometteuses apparaissent suite aux travaux réalisés et aux coopérations qui y sont liées. Les principales voies de développement possibles concernent l'intégration et la validation de l'estimateur de mouvement configurable dans la chaîne de codage vidéo H.264/AVC ainsi que l'incorporation dans la plateforme de compression vidéo pour des applications basse résolution, de mesures temps réel de reconnaissance de formes, de suivi d'objets ou de détection d'événements. Dans ces applications, la qualité de la vidéo est dépendante de l'évolution de la scène vidéo elle-même, notamment les zones d'intérêts (apparition de l'objet) de l'image. Ainsi, la zone d'intérêt peut être un individu ou le visage d'un individu. Une fois celui-ci localisé dans la scène, il est possible de le suivre (tracking) mais aussi de compresser les données de manière spécifique

en fonction de la région afin, par exemple, de pouvoir le reconnaître (les données de la zone du visage étant moins compressées que le reste de la scène).

D'autres débouchés à plus court terme sont envisageables pour l'implantation de la partie subpixélique basée sur la deuxième unité d'interpolation proposée (traitement de quatre macro-blocs 4x4 simultanément) qui pourrait atteindre le ¼ de pixel. Il reste également à finaliser la description du codeur (profil de base) en CAL.

Annexe

La description en CAL d'acteur SAD

```
actor SAD (dynamicSZ1,dynamicSZ2,dynamicSZ3)    /* Déclaration des
      int (size=dynamicSZ3) addr_xi,            Entrées/sorties d'acteur SAD */
      int (size=dynamicSZ3) addr_yi,
      int (size=dynamicSZ3) number_matching,
      int (size=dynamicSZ2) pa,
      int (size=dynamicSZ2) pb,
      int (size=dynamicSZ2) pc,
      int (size=dynamicSZ2) pd,
      int (size=dynamicSZ2) swa,
      int (size=dynamicSZ2) swb,
      int (size=dynamicSZ2) swc,
      int (size=dynamicSZ2) swd
      ==>
      int toto,
      int (size=dynamicSZ3)
number_matching_t,
      int (size=dynamicSZ3) score,
      int (size=dynamicSZ3) addr_xt,
      int (size=dynamicSZ3) addr_yt,
      int (size=dynamicSZ1+1)pp1t,
      int (size=dynamicSZ1+1)pp2t,
      int (size=dynamicSZ1+4)Diff1t
      :

      int (size=dynamicSZ3) ad_xt;              /* Déclaration des variables
      int (size=dynamicSZ3) ad_yt;              globales */
      int (size=dynamicSZ1+1) pp1;
      int (size=dynamicSZ1+1) pp2;
      int (size=dynamicSZ1+1) pp3;
      int (size=dynamicSZ1+1) pp4;
      int (size=dynamicSZ1+1) pp5;
      int (size=dynamicSZ1+1) pp6;
      int (size=dynamicSZ1+1) pp7;
      int (size=dynamicSZ1+1) pp8;
      int (size=dynamicSZ1+1) pp9;
      int (size=dynamicSZ1+1) pp10;
      int (size=dynamicSZ1+1) pp11;
      int (size=dynamicSZ1+1) pp12;
      int (size=dynamicSZ1+1) pp13;
      int (size=dynamicSZ1+1) pp14;
      int (size=dynamicSZ1+1) pp15;
      int (size=dynamicSZ1+1) pp16;

      int (size=dynamicSZ1+1) pw1;
      int (size=dynamicSZ1+1) pw2;
      int (size=dynamicSZ1+1) pw3;
      int (size=dynamicSZ1+1) pw4;
      int (size=dynamicSZ1+1) pw5;
      int (size=dynamicSZ1+1) pw6;
      int (size=dynamicSZ1+1) pw7;
```

```
        int (size=dynamicSZ1+1) pw8;
        int (size=dynamicSZ1+1) pw9;
        int (size=dynamicSZ1+1) pw10;
        int (size=dynamicSZ1+1) pw11;
        int (size=dynamicSZ1+1) pw12;
        int (size=dynamicSZ1+1) pw13;
        int (size=dynamicSZ1+1) pw14;
        int (size=dynamicSZ1+1) pw15;
        int (size=dynamicSZ1+1) pw16;

int (size=dynamicSZ1+1) diff1 :=0;                  /* Initialisation des variables
int (size=dynamicSZ1+1) diff2 :=0;                  globales utilisées dans les
int (size=dynamicSZ1+1) diff3 :=0;                  corps des actions */
int (size=dynamicSZ1+1) diff4 :=0;
int (size=dynamicSZ1+1) diff5 :=0;
int (size=dynamicSZ1+1) diff6 :=0;
int (size=dynamicSZ1+1) diff7 :=0;
int (size=dynamicSZ1+1) diff8 :=0;
int (size=dynamicSZ1+1) diff9 :=0;
int (size=dynamicSZ1+1) diff10 :=0;
int (size=dynamicSZ1+1) diff11 :=0;
int (size=dynamicSZ1+1) diff12 :=0;
int (size=dynamicSZ1+1) diff13 :=0;
int (size=dynamicSZ1+1) diff14 :=0;
int (size=dynamicSZ1+1) diff15 :=0;
int (size=dynamicSZ1+1) diff16 :=0;
int (size=dynamicSZ1+2) add11 :=0;
int (size=dynamicSZ1+2) add12 :=0;
int (size=dynamicSZ1+2) add13 :=0;
int (size=dynamicSZ1+2) add14 :=0;
int (size=dynamicSZ1+2) add15 :=0;
int (size=dynamicSZ1+2) add16 :=0;
int (size=dynamicSZ1+2) add17 :=0;
int (size=dynamicSZ1+2) add18 :=0;
int (size=dynamicSZ1+3) add21 :=0;
int (size=dynamicSZ1+3) add22 :=0;
int (size=dynamicSZ1+3) add23 :=0;
int (size=dynamicSZ1+3) add24 :=0;
int (size=dynamicSZ1+4) add31 :=0;
int (size=dynamicSZ1+4) add32 :=0;
int (size=dynamicSZ3+1) mad_temp3:= 0;
int (size=dynamicSZ3+1) mad_temp2:= 0;
int (size=dynamicSZ1+5) mad_temp1:= 0;
int (size=dynamicSZ3) nofm := 0;
bool end_process = true;
bool read_c := false;
bool fire := false;
bool pipeline_set := false;
bool k := false;
bool tok := false;
int (size=dynamicSZ1 )i := 0;
int (size=dynamicSZ1 )j := 0;
int toto_t;
int(size=dynamicSZ1)
index_o_temp;

                                                    /* début d'action mad1 */
mad1: action  number_matching: [nofm_temp]
```

```
==> number_matching_t :[nofm_temp]
      guard end_process = true
      do
            toto_t := 0;
            nofm := nofm_temp;
            read_c := true;
            tok := true;
            end_process := false;
            j := 0;i := 0;

pp1:=0;pp2:=0;pp3:=0;pp4:=0;pp5:=0;pp6:=0;
pp7:=0;pp8:=0;pp9:=0;pp10:=0;pp11:=0;pp12:=0;
pp13:=0;pp14:=0;  pp15:=0;pp16:=0;

pw1:=0;pw2:=0;pw3:=0;pw4:=0;pw5:=0;pw6:=0;
pw7:=0;pw8:=0;pw9:=0;pw10:=0;pw11:=0;pw12:=0;
pw13:=0;pw14:=0;pw15:=0;pw16:=0;

diff1:=0;diff2:=0;diff3:=0;diff4 :=0;
diff5:=0;diff6:=0;diff7:=0;diff8 :=0;
diff9:=0;diff10:=0;diff11 :=0;
diff12:=0;diff13:=0;diff14 :=0;diff15:=0;
diff16:=0;

add11 :=0;add12 :=0;add13 :=0;add14 :=0;
add15 :=0;add16 :=0;add17 :=0;add18 :=0;

add21 :=0;add22 :=0;add23 :=0;add24 :=0;

add31 :=0;add32 :=0;

mad_temp3:= 0;
mad_temp2:= 0;
mad_temp1:= 0;
end                                                  /* Fin d'action mad1 */

read_only: action pa :[pat], pb :[pbt], pc           /* début d'action read_only */
:[pct], pd :[pdt],  swa :[swat], swb :[swbt],
swc :[swct], swd :[swdt] ==>                         /* l'action read_only reçoit
pp1t:[pp1],pp2t:[pw1]                                deux lignes de 16 pixels
      guard read_c=true, tok=true                    chacune afin de dissocier
      do                                             chaque pixel appart.

pp1 := bitand(pat,255);                              Cette action est déclenchée
pp2 := bitand(rshift(pat,8),255);                    sauf si les deux drapeaux
pp3 := bitand(rshift(pat,16),255);                   « read_c » et « tok » sont
pp4 := bitand(rshift(pat,24),255);                   valides
pp5 := bitand(pbt,255);                              */
pp6 := bitand(rshift(pbt,8),255);
pp7 := bitand(rshift(pbt,16),255);
pp8 := bitand(rshift(pbt,24),255);
pp9 := bitand(pct,255);
pp10 :=bitand(rshift(pct,8),255);
pp11 := bitand(rshift(pct,16),255);
pp12 := bitand(rshift(pct,24),255);
pp13 := bitand(pdt,255);
pp14 := bitand(rshift(pdt,8),255);
```

```
pp15 := bitand(rshift(pdt,16),255);
pp16 := bitand(rshift(pdt,24),255);

pw1 := bitand(swat,255);
pw2 := bitand(rshift(swat,8),255);
pw3 := bitand(rshift(swat,16),255);
pw4 := bitand(rshift(swat,24),255);
pw5 := bitand(swbt,255);
pw6 := bitand(rshift(swbt,8),255);
pw7 := bitand(rshift(swbt,16),255);
pw8 := bitand(rshift(swbt,24),255);
pw9 := bitand(swct,255);
pw10 := bitand(rshift(swct,8),255);
pw11 := bitand(rshift(swct,16),255);
pw12 := bitand(rshift(swct,24),255);
pw13 := bitand(swdt,255);
pw14 := bitand(rshift(swdt,8),255);
pw15 := bitand(rshift(swdt,16),255);
pw16 := bitand(rshift(swdt,24),255);

tok := false;
pipeline_set := true;
end                                          /* Fin d'action mad1 */

mad_fire1:  action ==>                       /* Début d'action mad_fire1 */
      guard (pipeline_set = true)            /* il est Claire que cette
      do                                     action ni consomme ni produit
                                             les jetons, elle permet
      pipeline_set := false;                 d'exploiter les pixels
      if (i < 15)then                        dissociés dans l'action
      mad_temp2 := mad_temp2 + mad_temp1;    précédente afin de faire le
      else                                   calcul demandé.
      mad_temp3 := mad_temp2 + mad_temp1;    Ce calcul n'est que la Valeur
      mad_temp2 := 0;                        SAD en pipeline.
      end;

                                             Pour cela on voit qu'à la fin
      mad_temp1 := add31 + add32;            un test sur la variable 'j'
                                             permettant de remplir les
      add31 := add21 + add22;                étages pipelines avec les
      add32 := add23 + add24;                pixels reçus par l'action
                                             précédente avant de passer pour
      add21 := add11 + add12;                l'une des action suivantes.
      add22 := add13 + add14;                Comme toutes les actions,
      add23 := add15 + add16;                « mad_fire1 » est active avec
      add24 := add17 + add18;                la condition de « Guard »

      add11 := diff1 + diff2;                */
      add12 := diff3 + diff4;
      add13 := diff5 + diff6;
      add14 := diff7 + diff8;
      add15 := diff9 + diff10;
      add16 := diff11 + diff12;
      add17 := diff13 + diff14;
      add18 := diff15 + diff16;

if (pp1 > pw1) then diff1:= (pp1 - pw1); else
diff1:= (pw1 - pp1); end
```

```
if (pp2 > pw2) then diff2:= (pp2 - pw2); else
diff2:= (pw2 - pp2); end
if (pp3 > pw3) then diff3:= (pp3 - pw3); else
diff3:= (pw3 - pp3); end
if (pp4 > pw4) then diff4:= (pp4 - pw4); else
diff4:= (pw4 - pp4); end
if (pp5 > pw5) then diff5:= (pp5 - pw5); else
diff5:= (pw5 - pp5); end
if (pp6 > pw6) then diff6:= (pp6 - pw6); else
diff6:= (pw6 - pp6); end
if (pp7 > pw7) then diff7:= (pp7 - pw7); else
diff7:= (pw7 - pp7); end
if (pp8 > pw8) then diff8:= (pp8 - pw8); else
diff8:= (pw8 - pp8); end
if (pp9 > pw9) then diff9:= (pp9 - pw9); else
diff9:= (pw9 - pp9); end
if (pp10 > pw10) then diff10:= (pp10 - pw10);
else diff10:= (pw10 - pp10); end
if (pp11 > pw11) then diff11:= (pp11 - pw11);
else diff11:= (pw11 - pp11); end
if (pp12 > pw12) then diff12:= (pp12 - pw12);
else diff12:= (pw12 - pp12); end
if (pp13 > pw13) then diff13:= (pp13 - pw13);
else diff13:= (pw13 - pp13); end
if (pp14 > pw14) then diff14:= (pp14 - pw14);
else diff14:= (pw14 - pp14); end
if (pp15 > pw15) then diff15:= (pp15 - pw15);
else diff15:= (pw15 - pp15); end
if (pp16 > pw16) then diff16:= (pp16 - pw16);
else diff16:= (pw16 - pp16); end

if (j = 5) then fire:= true;
else j := j +1; tok := true;end
end                                          /* Fin d'action mad_fire1 */

mad_fire2: action ==> toto:[toto_t]          /* Début d'action mad_fire2 */
      guard fire= true ,(nofm > 0 and i < 15)
      do                                     /*Cette action accumule les
                                             valeurs de SAD avec des
      fire := false;                         conditions bien définies avec
      if (nofm = 1 and i > 9)then            « Guard »
      read_c := false;                       */
      pipeline_set := true;
      end
      i := i +1;
end                                          /* Fin d'action mad_fire2 */

mad_fire3: action addr_xi:[xi], addr_yi:[yi] /* Début d'action mad_fire3 */
                  ==>                        /*Cette action accumule la
score :[mad_temp3],addr_xt:[xi], addr_yt:[yi] dernière valeur pour les 16
guard fire= true , (nofm > 1 and i = 15)     èmes lignes de chaque
      do                                     macroblocs appart le dernier
                                             Afin de produire la valeur SAD
      ad_xt:=xi;                             associée à l'adresse de chaque
      ad_yt:=yi;                             macrobloc
      fire := false;
      tok := true;                           */
```

```
      i := 0;
            nofm := nofm -1;
      end                                          /* Fin d'action mad_fire2 */


mad_fire4: action addr_xi:[xi], addr_yi:[yi]       /* Début d'action mad_fire3 */
            ==>
            score :[mad_temp3],addr_xt:[xi],       /*Cette action accumule les
addr_yt:[yi]                                       deux dernières lignes du
guard fire= true ,(nofm = 1 and i = 15)            dernier macrobloc de haque
      do                                           phase en produisant sa valeur
      ad_xt:=xi;                                   SAD associée et son adresse
      ad_yt:=yi;
      fire:= false;
      pipeline_set := false;
      end_process := true;
      tok := false;
      i := 0;
      j := 0;
      nofm := nofm -1;
end                                                /* Fin d'action mad_fire4 */

 priority                                          /* La priorité ordonnance les
      mad1 > read_only;                            actions */
      mad1 > mad_fire1;
      mad1 > mad_fire2;
      mad1 > mad_fire3;
      mad1 > mad_fire4;
  end                                              /* Fin d'action mad_fire2 */


end                                                /* Fin d'acteur */
```

Glossaire

AVC	Advanced Video Coding
ADSL	Asymetric Digital Subscriber Line
ASIC	Application Description Language
BBGDS	Block-Based Gradient Descent Search
CABAC	Context Adaptive Binary Arithmetic Coding
CAVALC	Context Adaptive Variable Length Coding
CAL	CAL Actor Language
C_b	Chrominance bleu
C_r	Chrominance rouge
CIF	Common Intermediate Format
CRT	Cathode Ray Tube
CS	Cross Search
CDS	Cross Diamond Search
Db	Decibel
DCT	Discrete Cosine Transform
DS	Diamond Search
DVD	Digital Video Disk
DWT	Discrete Wavelet Transform
DPN	Data Process Networks
DSP	Digital Signal Processing
EMP	Estimation de Mouvement Pixélique
EMS	Estimation de Mouvement Subpixélique
EPZS	Enhanced Predictive Zonal Search
FBSME	Fixed Block-Size Motion Estimation
FIR	Finite Impulse Response
FME	Fractional Motion Estimation
FPGA	Field Programmable Gate Array
FS	Full Search
Gbits/s	Giga bits par seconde
Go	Giga octet
GOP	Group Of Pictures
HDTV	High Definition Television
HEVC	high Efficiency Video Coding
HEXBS	HEXAGON BASED SEARCH
ISO-IEC	International Organization for Standardization/International Electrotechnical Commission
JPEG	Joint Photographic Experts Groupe
KPN	Kahn Process Networks
LCD	Liquid Crystal Display
LDSP	Large Diamond Search Pattern
LED	Light Emitting Diode
MAD	Mean Absolute Difference
MB	Macrobloc
Mbits/s	Méga bits par seconde
MC	Motion Compensation
MDS	Modified Diamond Search

ME	Motion Estimation
MPDS	Multi-Point Diamond Search
MPEG	Moving Picture Experts Group
MSE	Mean Square Error
MV	Motion Vector
NTSC	National Television System Committee
NTSS	New Three Step Search
OpenDf	Open Data Flow
ORCC	Open RVC CAL Compiler
QCIF	Quarter Common Intermediate Format
QFHD	Quad Full High-Definition Resolution
PAL	Phase Alternating Line
PE	Processing Element
PMVFAST	Predictive Motion Vector Field Adaptive Search Technique
PSNR	Peak Signal to Noise Ratio
PU	Processing Unit
RVC	Reconfigurable Video Coding
SAD	Sum of Absolute Differences
SATD	Sum of Absolute Hadamard Transformed Differences
SDTV	Standard Definition Television
SDSP	Small Diamond Search Pattern
SECAM	Système Electronique Couleur avec Memoire
SMPTE	Society of Motion Picture and Television Engineers
SSD	Sum of Square Difference
TSS	Three Step Search
TTM	Time To Market
UHDTV	Ultra High Definition Television
UIT-T	Union Internationale des Télécommunications - Télécommunications
VBSME	Variable Block Size Motion Estimateion
VCEG	Video Coding Experts Group
VHDL	Very high speed integrated circuit Hardware Description Language
VLC	Variable Length Coding
VLSI	Very Large-Scale Integration
VoIP	Voice over Internet Protocol
4SS	Four Step Search

Liste des Figures

Liste des Tableaux

Bibliographie

[**AMH+11**] H. Aman-Allah, K. Maarouf, E. Hanna, I. Amer, M. Mattavelli, "*CAL Dataflow Components for an MPEG RVC AVC Baseline Encoder*", Journal of Signal Processing Systems for Signal, Image and Video Technology, Special Issue on Reconfigurable Video Coding, vol. 63, pp. 227–239, 2011.

[**ANR74**] N. Ahmed, T. Natarajan, K.R. Rao, "*Discrete Cosine Transform*", IEEE Trans. On Computer, vol. 23(1), Janvier, pp. 90-93; 1974.

[**BAIL1**] D.G. Bailey, "*Design For Embedded Image Processing on FPGAs*", John Wiley and Son, 2011.

[**BOU05**] S. Bouchoux, "*Apport de la reconfiguration dynamique au traitement d'images embarqué : étude de cas : implantation du d´encodeur entropique de JPEG 2000*", Thèse de l'Université de Bourgogne, 2005.

[**BWX11**] J. Bankoski, P. Wilkins, Y. Xu, "*Technical overview of VP8, an open source video codec for the web*", IEEE International Conference on Multimedia and Expo (ICME), pp. 1-6, juillet 2011.

[**CCC+08**] Y.H. Chen, T.C. Chen, S.Y. Chien, Y.W. Huang, L.G. Chen, "*VLSI architecture design of fractional motion estimation for H.264/AVC*", Journal of Signal Processing Systems, vol. 53(3), pp. 335–347, 2008.

[**CCH+06**] C-Y. Chen, S-Y. Chien, Y-W. Huang, T-C. Chen, T-C. Wang, L-G. Chen, "*Analysis and Architecture Design of Variable Block-Size Motion Estimation for H.264/AVC*", IEEE Transactions on Circuits and System-I: Regular Papers, vol. 53(2), février, pp. 578-593, 2006.

[**CHC04**] T.C. Chen, Y.W. Huang, L.G. Chen, "*Fully utilized and reusable architecture for fractional motion estimation of H.264/AVC*", In: IEEE ICASSP, pp. 9–12, 2004.

[**CP02**] C. Cheung, L.M. Po, "*A novel cross-diamond search algorithm for fast block motion estimation*", IEEE Transactions on Circuits and Systems for Video Technology, vol. 12(12), pp. 1168–1177, 2002.

[**DGL+11**] F. De Simone, L. Goldmann, J. S. Lee, T. Ebrahimi, "*Performance analysis of VP8 image and video compression based on subjective evaluations*", SPIE Optics and Photonics, Applications of Digital Image Processing XXXIV, Vol. 8135, 2011.

[**DTM+09**] J. Dubois, R. Thavot, R. Mosqueron, J. Miteran, C. Lucarz, "*Motion Estimation Accelerator with User Search Strategy in an RVC Context*", Proceedings of IEEE

International Conference On Image Processing (ICIP09), Cairo, Egypte, pp. 761-764, Novembre 2009.

[**DY11**] Y. Ding, X. Yan, "*A robust motion estimation with center-biased diamond search and its parallel architecture for motion-compensated de-interlace*", Journal of Supercomputing, Vol. 58, pp. 68–83, 2011.

[**EDA+11**]W. Elhamzi, J. Dubois, M. Atri, T. Richard, J. Gorin, J. Miteran, R. Tourki, "*An Efficient Hardware Implementation of Diamond Search Motion Estimation Based on CAL Dataflow Language*", In International Conference on Microelectronics (ICM'2011), pp. 1-6, 2011.

[**EDM+12**] W. Elhamzi, J. Dubois, J. Miteran, M. Atri, *"An efficient low-cost FPGA implementation of a configurable motion estimation for H.264 video coding",* Journal of Real-Time Image Processing, pp. 1-12, 2012.

[**EJ03**] J. Eker, J. Janneck, "*CAL Language Report*", Technical Report ERL Technical Memo UCB/ERL M03/48, University of California at Berkeley, December 2003.

[**FAR08**] N. Farrugia, "*Architectures parallèles pour l'analyse de visages embarquée*", Thèse de l'Université de Bourgogne, 2008.

[**FAS09**] M.R.H. Fatemi, H.F. Ates, R. Salleh, "*A bit-serial sum of absolute difference accelerator for variable block size motion estimation of H.264*", In: Proceedings of of the Conference on Innovative in Intelligent Systems and Industrial Applications, pp. 1–4, 2009.

[**GAU**] http://hls-labsticc.univ-ubs.fr/

[**Gha90**] M. GHANBARI, "*The Cross-Search Algorithm for Motion Estimation*", IEEE Transactions on Circuits and Systems for Video Technology, vol. 38(7), pp. 950–953, 1990.

[**GHJ05**] W. Gao, M-Z. Hu, Z.Z. Ji, "*An efficient hardware implementation for motion estimation of AVC standard*", IEEE Transactions on Consumer Electronics, vol. 51(4), pp. 1360–1366, 2005.

[**GKL01**] L. Guan, S-Y. Kung, J. Larsen, "*Multimedia Image and Video Processing*", CRC Press, 2001.

[**HJK+03**] M. Horowitz, A. Joch, F. Kossentini, and A. Hallapuro, "*H.264/avc baseline profile decoder complexity analysis*", IEEE Transactions on Circuits and Systems for Video Technology, vol. 13, pp. 704 – 716, juillet 2003.

[**HL92**] C-H. Hsieh, T-P. Lin, "*VLSI Architecture for Block-Matching Motion Estimation Algorithm*", IEEE Transactions on Circuits and Systems for Video Technology, vol. 2(2), pp. 169-175, juin 1992.

[**HYP+97**] L. Haas, F. Yang, M. Paindavoine, C. Milan, "*Adéquation de l'algorithme de Canny-Deriche généralisé sur architecture DSP avec l'environnement SynDEx*", Traitement du Signal - Volume 14 - n°6 – Spécial, pp 625-635, 1997.

[**IEE05**] IEEE Std 1666-2005, IEEE Std 1666, "*IEEE Standard SystemC Language, Reference Manual*", 2005.

[**IMS+09**] Y. Ismail, J. McNeelly, M. Shaaban, M. Bayoumi, "*Enhanced efficient diamond search algorithm for fast block motion estimation*", In: IEEE ISCAS, pp. 3198–3201, Taipei, 2009.

[**ISO91**] ISO/IEC 11172-2, "*Standard mpeg-1 : coding of moving pictures and associated audio for digital storage media at up to about 1,5 mbits/s*", tech. rep., 1991.

[**ISO93**] "*ITU-T Recommendation H.261: Video Codec for Audiovisual Services at px64 kbits/s*", tech. rep., 1993.

[**ISO94**] ISO/IEC 13818-2, "*Standard mpeg-2: information technology-generic coding of moving pictures and associated audio information*", tech. rep., 1994.

[**ISO95**] ISO/IEC 13818-2, "*Recommendation ITU-T H.262 (1995 E)*", 1995.

[**ISO96**] "*Video Coding for Low Bit rate Communication*", tech. rep., 1996.

[**ISO98**] ISO/IEC 14496-2, "*Standard mpeg-4: information technology-generic coding of audio-visual objects*", tech. rep., 1998.

[**ISO02a**] ISO/IEC 15444-3, "*Information technology- JPEG 2000 image coding system-Part 3: Motion JPEG 2000*", 2002.

[**ISO02b**] DRAFT ITU-T Rec. H.264 (2002 E) "*Editor's Proposed Draft Text Modifications for Joint Video Specification (ITU-T Rec. H.264 |ISO/IEC 14496-10 AVC), Draft 7*", 5th Meeting: Geneva, Switzerland, 9-17 Octobre, 2002.

[**ISO11**] ISO/IEC/JTC1/SC29/WG11 N11872, "*Vision, applications and requirements for high efficiency video coding (HEVC)*", Daegu, South Korea, Janvier 2011.

[**ITU**] ITU, http://www.itu.int/

[**JMP+08a**] J.W. Janneck, I.D. Miller, D.B. Parlour, G. Roquier, M. Wipliez, M. Raulet, "*Synthesizing Hardware from Dataflow programs: an Mpeg-4 Simple Profile Decoder Case Study*", IEEE Workshop on Signal Processing Systems, Washington : United States, pp. 287-292, 2008.

[**JMP+08b**] J. Janneck, I. Miller, D. Parlour, M. Mattavelli, C. Lucarz, M. Wipliez, M. Raulet, G. Roquier, "*Translating dataflow programs to efficient hardware : an MPEG-4 Simple Profile decoder case study*", In Design, Automation and Test in Europe conference (DATE 2008), Munich, Allemagne, pp. 799, 2008.

[**JMP+11**] J.W. Janneck, I.D. Miller, D.B. Parlour, G. Roquier, M. Wipliez, M. Raulet, "Synthesizing Hardware from Dataflow Programs", *Journal of Signal Processing Systems*, vol. 36, pp. 241-249, 2011.

[**KIH+81**] T. Koga, K. Ilinuma, A. Hirano, Y. Iijima, T. Ishiguro, "*Motion compensated interframe coding for video conferencing*", In: Proc.Nat. Telecommun Conf, pp. G5.3.1–G5.3.5, New Orleans, 1981.

[**KLA+11**] M. Kthiri, H. Loukil, A.B. Atitallah, P. Kadionik, D. Dallet, N. Masmoudi "*FPGA architecture of the LDPS Motion Estimation for H.264/AVC Video Coding*", Journal of Signal Processing System, vol. 68(2), pp. 273-295, 2011.

[**KP89**] T. Komarek, P. Pirsch, "*Array architectures for block matching algorithms*", IEEE Transactions on Circuits and Systems for Video Technology, vol. 36(10): pp.1301–1308, 1989.

[**LF96**] L. Liu, E. Feig, "*A block-based gradient descent search algorithm for block motion estimation in video coding*", IEEE Transactions on Circuits and Systems for Video Technology, vol. 6(4), pp. 419–422, 1996.

[**LJL+03**] P. List, A. Joch, J. Lainema, G. Bjontegaard, and M. Karczewicz, "*Adaptive Deblocking filter*", IEEE Transactions on Circuits and Systems for Video Technology, vol. 13 (7), pp. 614-619, July 2003.

[**LKK05**] S-J. Lee, C-G. Kim, S-D. Kim, "*A pipelined hardware architecture for motion estimation of H.264/AVC*", In: Proceedings of the 10th Asia-Pacific conference on Advances in Computer Systems, Architecture ACSAC'05, pp. 79–89, 2005.

[**LKK+10**] Y-L.S. Lin, C-Y. Kao, H-C. Kuo, J-W. Chen, "*VLSI Design for Video Coding: H.264/AVC Encoding from Standard Specification to Chip*", Springer, 2010.

[**LL08**] B.M. H. Li, P.H.W. Leong, "*Serial and Parallel FPGA-based Variable Block Size Motion Estimation Processors*", Journal of Signal Processing Systems 51, pp. 77–98, 2008.

[**LMW+08**] C. Lucarz, M. Mattavelli, M. Wipliez, G. Roquier, M. Raulet, J. Janneck, "*Dataflow/Actor-oriented language for the design of complex signal processing systems*", Conference on Design and Architectures for Signal and Image Processing (DASIP 2008), pp 168-175, 2008.

[**LR06**] Y.G. Lee, J.B. Ra, "*Fast motion estimation robust to random motions based on a distance prediction*", IEEE Transactions on Circuits and Systems for Video Technology, vol. 16(7), pp. 869–875, 2006.

[**LT03**] A. Luthra, P. Topiwala "*Overview of the H.264/AVC video coding standard*", *Proc.* SPIE 5203, Applications of Digital Image Processing XXVI, pp. 560-576, Novembre 2003.

[**LZL94**] R. Li, B. Zeng, L.M. Liou, "*A new three-step search algorithm for fast motion estimation*", IEEE Transactions on Circuits and Systems for Video Technology, vol. 4(4), pp. 438–442, 1994.

[**MGS+06**] H. Mahmoud, S. Goel, M. Shaaban, M. Bayoumi "*A New Efficient Block-Matching Algorithm for Motion Estimation*", Journal of VLSI Signal Processing, vol. 42, pp. 21–33, 2006.

[**MPEG**] MPEG, http://www.mpeg.org/

[**MW10**] M. Wipliez, "*Infrastructure de compilation pour des programmes flux de données*", Thèse de l'INSA de Rennes, 2010.

[**NO11**] O. Ndili, T. Ogunfunmi, "*Algorithm and Architecture Co-Design of Hardware-Oriented, Modified Diamond Search for Fast Motion Estimation in H.264/AVC*", IEEE Transactions on Circuits and Systems for Video Technology, vol. 21 (9), pp. 1214-1227, Septembre 2011.

[**NR80**] A. N. Netravali, J. D. Robbins. "*Motion compensated coding: Some results*", The Bell System Technical Journal, vol 59(9), pp 1735–1745, Nov 1980.

[**OBL+04**] J. Ostermann, J. Bormans, P. List, D. Marpe, M. Narroschke, F. Pereira, T. Stockhammer, T. Wedi "*Video coding with H.264/AVC: Tools, Performance, and Complexity*", IEEE Circuits And Systems Magazine, vol. 4(1), pp. 7-28, 2004.

[**OMG09**] Object Management Group. – Unified modeling language, version 2.2, 2009. http://www.omgmarte.org.

[**PM96**] L.M. Po, W.C. Ma, "*A novel four-step search algorithm for fast block motion estimation*", IEEE Transactions on Circuits and Systems for Video Technology, vol. 6(3), pp. 313–317, 1996.

[**Ric02**] Iain E. Richardson, "*H.264 / MPEG-4 Part 10: Transform & Quantization", h.264/mpeg-4 part 10 white paper*", tech. rep., 2002.

[**Ric10**] Iain E. Richardson, "*The H.264 Advanced Video Compression Standard*", John Wiley and Sons, second edition, 2010.

[**RM10**] G.A. Ruiz, J.A. Michell, "*An efficient VLSI architecture of fractional motion estimation in H.264 for HDTV*", Journal of Signal Processing Systems, vol. 62(3), pp. 443–457, 2010.

[**RS02**] N. Roma, L. Sousa, "*Efficient and Configurable Full-Search Block-Matching Processors*", IEEE Transactions on Circuits and Systems for Video Technology, vol. 12(12), pp 1160-1167, Décembre 2002.

[**RWR+08**] G. Roquier, M. Wipliez, M. Raulet, J.W. Janneck, I.D. Miller, D.B. Parlour, "*Automatic software synthesis of dataflow program : An MPEG-4 simple profile*

decoder case study", In IEEE workshop on Signal Processing Systems (SiPS), pp. 281-286, 2008.

[**SLI+06**] Y. Song, Z. Liu, T. Ikenaga, S. Goto, "*A VLSI architecture for variable block size motion estimation in H.264/AVC with low cost memory organization*", IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences, vol. E89(12), pp. 3594–3601, 2006.

[**SNP+11**] G. Sanchez, D. Noble, M. Porto, S. Bampi, L. Agostini "*A Real Time HDTV Motion Estimation Architecture for the New MPDS Algorithm*", EUROCON - International Conference on Computer as a Tool, pp 1-4, avril 2011.

[**STL04**] G.J. Sullivan, P. Topiwala, A. Luthra, *"The H.264/AVC Advanced Video Coding* Standard*: Overview and Introduction to the Fidelity Range Extensions"*, SPIE conference on Applications of Digital Image Processing XXVII, 2004.

[**SYN**] SynDex, http://www.syndex.org/

[**TAL01a**] A.M. Tourapis, O.C. Au, M.L. Liou, "*Predictive Motion Vector Field Adaptive Search Technique (PMVFAST) Enhancing Block Based Motion Estimation*", In Visual Communications and Image Processing, pp. 883-892, 2001.

[**TAL01b**] A. M. Tourapis, O. C. Au, M. L. Liou, "*New results on zonal based motion estimation algorithms - advanced predictive diamond zonal search*", In IEEE International Symposium on Circuits and Systems, vol. 5, pp. 183–186, May 2001.

[**TC11**] N.T. Ta, J.R. Choi, "*High performance fractional motion estimation in h.264/avc based on one-step algorithm and 8x4 element block processing*", *SignalProcessing: Image Communication*, vol. 26, pp. 85–92, 2011.

[**TMD+09**] R. Thavot, R. Mosqueron, J. Dubois, M. Mattavelli, "*Hardware synthesis ofcomplex standard interfaces using cal dataflow descriptions*", Workshop on Design and Architectures for Signal and Image Processing (DASIP09), Sophia Antipolis, France, September 2009.

[**TMD+11**] R. Thavot, R. Mosqueron, J. Dubois, M. Mattavelli, "*Generation of Hardware/Software systems based on CAL dataflow description*", Best papers from Design and Architectures for Signal and Image Processing 2007 & 2008 & 2009, Series Lectures Notes in Electrical Engineering, vol. 73, Springer Verlag, pp. 275-292, Janvier 2011.

[**TMK00**] A. Takagit, S. Muramatsutt, H. Kiyat, "*Motion Estimation with Power Scalability and its VHDL Model*", International Conference on Image Processing, vol. 3, pp 118-121, 2000.

[**Tou02**] A.M. Tourapis, "*Enhanced Predictive Zonal Search for Single and Multiple Frame Motion Estimation*", Proceedings of Visual Communications and Image Processing, pp. 1069-1079, 2002.

[**UPN+72**] F. Urban, R. Poullaouec, J-F. Nezan, O. Deforges, "*A flexible heterogeneous hardware/software solution for real-time high-definition H.264 motion estimation*", IEEE Transactions on Circuits and Systems for Video Technology, vol. 18(12), pp. 1781-1785, Dec. 2008.

[**URB07**] F. Urban, "Implantation optimisée d'estimateurs de mouvement pour la compression vidéo sur plates-formes hétérogènes multicomposants", Thèse de l'INSA de Rennes 2007.

[**VS89**] L. Vos, M. Stegherr, "*Parameterizable VLSI architectures for the full-search block matching algorithm*", IEEE Trans Circuits Syst., vol. 36(10): pp. 1309–1316, 1989.

[**Wie03**] M. Wien, "*Variable Block-Size Transforms for H.264/AVC*", IEEE Transactions on Circuits and Systems for Video Technology, vol. 13(7), pp. 604-613, Juillet 2003.

[**WOO05**] C. Wootton, "*A Practical Guide to Video and Audio Compression*", Elsevier Inc, 2005.

[**WOS+10**] T. Wiegand, J. R. Ohm, G. J. Sullivan, W. J. Han, R. Joshi, T. K. Tan, K. Ugur, "*Special section on the joint call for proposals on high efficiency video coding (HEVC) standardization*", IEEE Transactions on Circuits and Systems for Video Technology, vol. 20 (12), pp. 1661-1666, Dec. 2010.

[**WP03**] T. Wieg, E. Pattaya, "*Draft ITU-T Recommendation H.264 and Draft ISO/IEC 14496-10 AVC*". In JVC of ISO/IEC and ITU-T SG16/Q.6 Doc. JVT-G050, Mar. 2003.

[**WSB+03**] T. Wiegand, G.J. Sullivan, G. Bjontegaard, A. Luthra, "*Overview of the H.264 / AVC Video Coding Standard*", IEEE Transactions On Circuits And Systems For Video Technology, vol. 13(7), pp. 560-576, Juillet 2003.

[**XCY03**] J. Xu, Z. Chen, Y. He, "*Efficient fast ME predictions and early-termination strategy based on H.264 statistical characters*", In Proceedings of the Joint Conference of the International Conference on Information, Communications and Signal Processing and Pacific-Rim Conference on Multimedia, pp. 218-222, 2003.

[**YGI06**] C. Yang, S. Goto, T. Ikenaga, "*High performance VLSI architecture of fractional motion estimation in H.264 for HDTV*", In Proceedings of the IEEE ISCAS, pp. 2605–2608, Greece, 2006.

[**YH95**] H. Yeo, Y-H. Hu, "*A novel modular systolic array architecture for full-search block matching motion estimation* IEEE Transactions On Circuits And Systems For Video Technology, vol. 5(5), pp 407-416, 1995.

[**YM04**] S.Y. Yap, J.V. McCanny, "*A VLSI architecture for variable block size video motion estimation*", IEEE Transactions on Circuits and Systems for Video Technology, vol. 51(7), pp. 384–389, 2004.

[**YSW89**] K-M. Yang, M-T. Sun, L. Wu, **"***A Family of VLSI Designs for the Motion Compensation Block-Matching Algorithm*", IEEE Transactions on Circuits and Systems, vol. 36 (10), Octobre 1989.

[**ZLC02**] C. Zhu, X. Lin, L.P. Chau, "*Hexagon-based search pattern for fast block motion estimation*", IEEE Transactions On Circuits And Systems For Video Technology, vol. 12(5), pp. 349–355, 2002.

[**ZLC+04**] C. Zhu, X. Lin, L-P. Chau, L-M. Po, "*Enhanced Hexagonal Search for Fast Block Motion Estimation*", IEEE Transactions On Circuits And Systems For Video Technology, vol. 14(10) : pp.1210–1214, 2004.

[**ZM00**] S. Zhu, K.K. Ma, "*A new diamond search algorithm for fast block matching motion estimation*", IEEE Transactions on Image Processing, vol. 9(2), pp. 287–290, 2000.

Résumé : L'objectif de cette thèse est la conception d'une plateforme de compression vidéo de nouvelle génération à haut degré d'adaptation vis-à-vis de l'environnement. Ce besoin d'adaptabilité a plusieurs origines. D'une part les systèmes actuels visent à s'adapter à la diversité et l'hétérogénéité des médias et des terminaux actuels. D'autre part, l'exploitation de l'information contenue dans une scène vidéo dépend de l'application visée et des besoins des utilisateurs. Ainsi, l'information peut être exploitée de manière complètement inhomogène spatialement ou temporellement. En effet, l'exploitation spatiale de la scène peut être irrégulière par définition, par la définition automatique ou manuelle de zones d'intérêts dans l'image. La qualité de la vidéo, donc de la compression, doit pouvoir s'adapter afin de limiter la quantité de donnée à transmettre. Cette qualité est donc dépendante de l'évolution de la scène vidéo elle-même. Une architecture matérielle configurable a été proposée dans cette thèse permettant de supporter différents algorithmes de recherche en offrant une précision subpixélique.

La synthèse des travaux menés dans ce domaine et la comparaison objective des résultats obtenus par rapport à l'état de l'art. L'architecture proposée est synthétisée à base d'un FPGA Virtex 6 FPGA, les résultats obtenus pourraient traiter l'estimation du mouvement pixélique avec un flux vidéo haute définition (HD 1080), respectivement à 13 images par seconde en utilisant la stratégie de recherche exhaustive (108K Macroblocs/s) et jusqu'à 223 images par seconde avec la recherche selon un grille en diamant (1,8 M Macroblocs /s). En outre le raffinement subpixélique en quart-pel est réalisé à Macroblocs 232k/ s.

Abstract: The aim of this thesis was to define and implement a hardware architecture of a configurable motion estimation capable of supporting various search strategies with the desired accuracy for adaptive video compression. This need for adaptability had several origins. Firstly, the current systems are designed to adapt to the diversity and heterogeneity of current terminals and media. Secondly, the use of information contained in a video scene depends on the intended applications and user needs. This objective scoring modestly in the challenge offered by the development of digital video requires a faster processing and a high compression ratio.

In this thesis, a flexible hardware implementation of the motion estimator which enables the integer motion search algorithms to be modified and the fractional search as well as variable block size to be selected and adjusted. Hence, this novel architecture, especially designed for FPGA targets, proposes high-speed processing for a configuration which supports the variable size blocks and quarter-pel refinement, as described in H.264. The proposed low-cost architecture based on Virtex 6 FPGA can process integer motion estimation on 1080 HD video streams, respectively, at 13 fps using full search strategy (108k Macroblocks/s) and up to 223 fps using diamond search (1.8M Macroblocks/s). Moreover subpel refinement in quarter-pel mode is performed at 232k Macroblocks/s.

SPIM

■ École doctorale SPIM - Université de Bourgogne/UFR ST BP 47870 F - 21078 Dijon cedex
■ tél. +33 (0)3 80 39 59 10 ■ ed-spim@univ-fcomte.fr ■ www.ed-spim.univ-fcomte.fr