Proposition et implémentation d'architectures matérielles pour les opérateurs utilisés dans les normes de compression d'images

Kamel Messaoudi[1, 2], Salah Toumi[1] & El-Bay Bourennane[1, 2]

[1] *Laboratoire d'Etude et de Recherche en Instrumentation et en Communication (LERICA)*
Université Badji Mokhtar Annaba, BP 12, 23000 Annaba, Algérie

[2] *Laboratoire d'Electronique, Informatique et Image (LE2I)*
Université de Bourgogne, BP 47 870, 21078 Dijon Cedex, France

Révisé le 03/12/2011
Accepté le 27/02/2012

ملخص
بالإضافة إلى التقنيات المتوازية المستعملة لقراءة أو تسجيل أو إعادة إستدعاء الماكروبلوك من الصور في حالات التطبيقات ذات الوقت الحقيقي وخوارزميات معالجة الصور، فإنه من الضروري كذلك ضمان تقنيات متوازية لوحدات الحسابات والتي عادة ما تكون تكرارية الصيغ . نفس التقنيات يجب أن تستعمل أيضا لضمان وحدة التزامن . في هذا العمل، نقدم حلولا تطبيقية جديدة لتنفيذ وحدات الحسابات المستخدمة في خوارزميات معالجة الصور ومقاطع الفيديو . النماذج التطبيقية المقترحة و المبنية على مبدأ التوازي، برمجت باستعمال برامج VHDLوتحقق منها على منصات تطبيقية من نوع شرائح FPGA .

الكلمات المفتاحية : *نماذج تطبيقية – وحدات الحسابات – مبدأ التوازي – SAD – نظام التشفير H.264* .

Résumé
En plus des techniques parallèles de lecture, d'enregistrement et d'appel des sous blocs d'images, pour assurer le temps réel pour les algorithmes de traitement des images, il est nécessaire aussi d'implanter des structures parallèles pour les modules de calcul (les opérateurs) qui sont généralement des formules itératives. L'ensemble des structures et techniques doivent être synchronisé par une unité de contrôle. Dans ce papier, nous présentons une nouvelle solution d'implémentation matérielle des opérateurs utilisés dans les algorithmes de traitement des images et des vidéos. Les implémentations proposées, basées sur le parallélisme des données, sont réalisées en VHDL et vérifiées sur des plateformes reconfigurables de type FPGA.

Mots clés *: Architecture matérielle – Opérateurs – Parallélisme des données – SAD – Encodeur H.264.*

Abstract
To ensure real-time applications using image processing algorithms, several parallel techniques are used to read, to store or to call the image macroblocks. It is also necessary to ensure parallel structures for the calculation modules (operators) which are generally iterative formulas, and also to ensure the synchronization of all modules. In this paper, we present a new solution of a hardware implementation of operators used in image and video algorithms. The proposed hardware implementations, based on the data parallelism principle, are realized in VHDL and verified on reconfigurable platforms like FPGAs.

Keywords *: Hardware architecture - Operators - Data parallelism - SAD - H.264 encoder.*

Auteur correspondant : kamel.messaoud@univ-annaba.org

1. INTRODUCTION

Pour répondre aux besoins du temps réel, d'un grand nombre d'applications multimédias, des solutions d'implémentations matérielles sur des plateformes reconfigurables de type FPGA (Field-Programmable Gate Array) ont été proposées dans plusieurs travaux [1]. En effet, le parallélisme des données et des instructions existe sous plusieurs formes dans les systèmes sur puce, de la première forme SISD (Single Instruction Single Data) qui est réellement une structure non parallèle, jusqu'à la dernière forme MIMD (Multiple Instruction Multiple Data). Cette dernière structure, basée sur plusieurs processeurs exécutant différentes instructions sur différentes données, est la plus efficace pour obtenir de vraies machines parallèles. En plus des structures des données dans un algorithme de traitement d'image, on trouve souvent des opérateurs mathématiques qui s'occupent principalement du temps de calcul d'un microprocesseur. Pour assurer un temps réel pour ce type d'applications, il est nécessaire de proposer des architectures matérielles pour accélérer le temps de calcul des opérateurs [2]. Dans le présent article, nous allons montrer comment exploiter au mieux les architectures parallèles pour l'implantation matérielle des opérateurs utilisés dans les algorithmes de traitement d'images. Nous allons également justifier nos choix sur les parties réalisées en matériel et les parties réalisées en logiciel dans une implémentation mixte nommée aussi codesign [3]. Dans ce travail, nous introduisons les notions des systèmes embarqués, des SoCs, des MPSoCs et le codesign (section 2). Nous justifions, ensuite dans la section 3, nos choix sur les parties réalisées en matériel et les parties réalisées en logiciel dans le cas particulier de la norme de compression vidéo H.264/MPEG4 part 10 [4].
Nous décrivons dans la section 4 les opérateurs de traitement d'image choisis. Des implémentations matérielles basées sur le principe de parallélisme des données sont présentées dans les deux sections 5 et 6. Il sera, finalement, proposé différentes utilisations des opérateurs implantés en matériels dans les applications de traitement de vidéo.

2. SYSTEME EMBARQUE, CODESIGN ET SOC

Un système embarqué (SE) peut être défini comme un système électronique et informatique autonome ne possédant pas des entrées/sorties standards comme un clavier ou un écran d'ordinateur. C'est un système autonome qui utilise généralement un processeur et exécute un logiciel dédié pour réaliser une fonctionnalité précise. Des circuits numériques et analogiques sont utilisés en plus pour augmenter les performances du SE ou sa fiabilité. Dans un SE des opérations de calcul doivent être faites en réponse à un évènement extérieur (interruption matérielle), ce qui signifie un fonctionnement en temps réel et en réactivité [5].

2.1 Le codesign

Le terme "Codesign = conception des systèmes mixtes logiciel/matériel" est apparu au début des années 1990 pour marquer une nouvelle façon de la conception des circuits intégrés et des systèmes numériques. La "conception conjointe du logiciel et du matériel" était devenue nécessaire pour répondre aux exigences du marché des systèmes intégrés. En effet, l'émergence des systèmes multimédia (téléphone portable, console de jeu, …) entraînait une plus grande complexité de la partie électronique et la concurrence économique imposait un temps de conception encore plus court.

2.2 Les SoCs et les SoPCs

Un SoC (System on Chip) est un ensemble de blocs fonctionnels intégrés dans un composant électronique avec au moins un processeur comme élément de traitement. L'idée des SoC consiste à implanter en matériel les blocs de traitement particulièrement coûteux de point de vue temps de calcul et l'essentiel de l'application étant implanté en logiciel et exécuté par les cœurs des processeurs. Ceci conduit à un autre problème de choix entre les blocs de traitement à réaliser en matériel et les blocs à réaliser en logiciel (Fig.1).

L'approche SoC a été créée dans un premier temps pour le développement d'ASIC. Ensuite cette approche a été étendue pour le développement des applications sur des plateformes reconfigurables de type FPGA. On parle alors de SoPC pour système sur puce programmable (System on Programmable Chip).

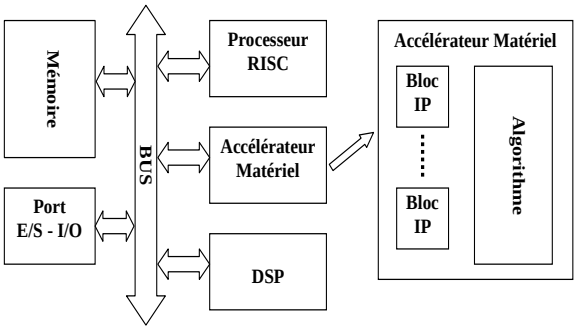

Figure 1. Structure d'un système sur puce SoC.

3. CHOIX DES PARTIES MATERIELLE ET LOGICIELLE

La conception des systèmes contenant des composants matériels et des composants logiciels (hardware/software codesign), doit être élaborée à partir d'une spécification unique décrivant son architecture et/ou son comportement. Après la phase de spécification survient une phase de partitionnement du système ayant pour but de décomposer ce dernier en partitions comportant trois parties (une partie matérielle, une partie logicielle et une interface de communication entre les deux premières parties). La partie matérielle absorbant les sections de l'algorithme exige un nombre de cycles d'horloge (temps de calcul) important. Dans un algorithme de traitement d'image, on trouve souvent des composants qui prennent la majorité du temps de calcul en utilisant un microprocesseur, ce qui les rend la première cible de matérialisation. Dans un exemple de compression de vidéo (la norme H.264/AVC nommée aussi MPEG4 part10) [6], la complexité est fortement localisée dans les trois modules de traitement de l'encodeur [7] :

- L'estimation et la compensation de mouvement EM/CM (60 à 70 % du temps CPU).
- La transformée en cosinus discrète directe et inverse TCD/TCDI (20 à 30 % du temps CPU).
- Le filtre anti-blocs ou bien le deblocking filter (autours de 10%).

Une étude plus fine sur le bloc d'estimation et de compensation de mouvement [3, 5] indique que les ressources sont principalement utilisées par l'opérateur de calcul de distorsion, à savoir, la somme des valeurs absolues des différences (SAE pour Sum of Absolute Errors). En effet, ce calcul de distorsion représente à lui seul 50 à 70% du temps CPU total pour l'encodeur H.264. Ces résultats permettent de faire une première hypothèse sur la répartition matérielle et logicielle des traitements dans le cas d'une implantation de l'encodeur H.264/AVC sur une plateforme reconfigurable.

4. LES OPERATEURS DE TRAITEMENT D'IMAGE

Les opérateurs de traitement d'image sont les modules de calcul de base qui constituent les algorithmes les plus évolués. Dans ce travail, nous allons montrer les intérêts d'application du principe de parallélisme dans le cas des trois opérateurs : le produit scalaire, le filtrage et l'opérateur SAE. Ces opérateurs sont très utilisés dans les algorithmes et les normes de compression vidéo y compris la norme H.264/AVC.

4.1 Le produit scalaire

Le produit scalaire, ou bien point à point, est sans doute l'application la plus simple à mettre en œuvre est l'une des opérations les plus récurrentes dans les algorithmes de traitement du signal et de l'image. Le produit scalaire prend comme argument deux vecteurs de même taille et donne en sortie un scalaire selon l'équation suivant [8] :

$$a.b = \sum_{i=0}^{N-1} a_i.b_i \qquad (1)$$

4.2 Les filtres

La majorité des algorithmes de traitement d'image en particulier les filtres numériques, utilisent des blocs (des fenêtres d'image) de taille N×N, pour avoir en sortie comme résultat un pixel, un scalaire, deux scalaires ou bien même un autre bloc de pixels. En effet, dans un filtre pour traiter chaque pixel de l'image courante, il faut connaître les autres pixels qui l'entourent dans la même image ou bien dans les autres images précédentes. Nous avons montré auparavant [5, 9] plusieurs techniques de sélection et d'adressage des blocs d'image dans les normes de compression. L'idée consiste à fixer la fenêtre de calcul et à déplacer les pixels désirés vers cette fenêtre par rotation et/ou décalage de pixels.
Prenons l'exemple des filtres linéaires suivants: moyen, gaussien, gradient vertical et gradient horizontal. Ces filtres traitent l'image en utilisant des fenêtres de 3×3 pixels avec

différents coefficients de pondération. Dans un filtre numérique à réponse impulsionnelle finie (FIR), étant données les valeurs des pixels couverts par la fenêtre appliquée (A_i avec i = 1 à 9), et les coefficients normalisés de cette fenêtre (C_i avec i = 1 à 9), le pixel central (filtré) est calculé par la formule suivante [3] :

$$PCF = \sum_{i=1}^{9} C_i . A_i \qquad (2)$$

Ou bien dans un espace à deux dimensions et avec une fenêtre de taille 3×3 :

$$PCF = \sum_{i=1}^{3}\sum_{j=1}^{3} C(i,j).A(i,j) \qquad (3)$$

Généralement les coefficients de pondération C(i,j) sont des +/-1 ou bien des zéros, ce qui rend l'opération plus simple sous forme de sommes et de soustractions.

4.3 L'opérateur SAE

L'opérateur SAE utilisé dans les normes de compression vidéo sert à la détection et à l'estimation de mouvement dans la méthode Block Matching. Cette méthode consiste à comparer tous les blocs de taille N×N dans l'image prédite 'P' avec les blocs de même taille dans l'image de référence 'I', l'opération de recherche est limitée dans une fenêtre de taille D×D à partir du centre du bloc central (Fig. 2).

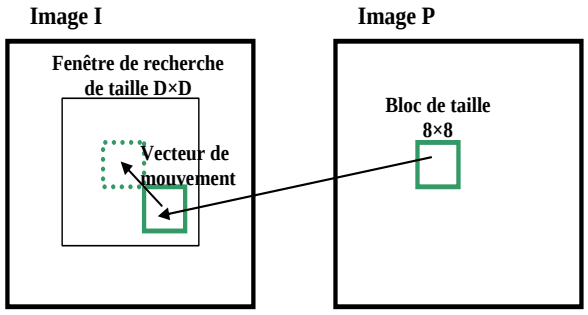

Figure 2. Principe d'estimation de mouvement (Block Maching).

La comparaison prend toutes les combinaisons possibles du bloc de recherche (Full search) avec l'inconvénient de quantité de calcul, ou bien quelques combinaisons pour réduire la quantité de calcul. En effet, plusieurs algorithmes rapides (3steps, 4steps, hexagon, ….) ont été proposés avec l'inconvénient de tomber dans un minimum local [10].
L'estimation de mouvement en utilisant la méthode Block Matching consiste à minimiser la différence entre un bloc central de l'image prédite 'P' et plusieurs blocs de l'image de référence 'I'. Dans notre cas, si on adopte une estimation de type full search, qui calcule toutes les SAE pour touts les déplacements possibles, et pour des images de taille M×K, l'algorithme sera le suivant [11] :

Pour i=1 à M/N faire

Pour j=1 à K/N faire

…..

Pour k =i-D/2 à i+D/2 faire

Pour l =j-D/2 à j+D/2 faire

$$SAE(k,l) = \sum_{r=0}^{7}\sum_{c=0}^{7} abs\left[I(i+k+r, j+l+c) - P(i+r, j+c)\right]$$

……

Fin l

Fin k

$$(k_{min}, l_{min}) = \underset{k,l}{\operatorname{argmin}}\ SAE(k,l)$$

……

Fin j

Fin i

Le même principe pour les autres opérateurs mathématiques, comme par exemple [8] :

- Mean Squared Error :

$$MSE(k,l) = \frac{1}{N^2}\sum_{r=0}^{7}\sum_{c=0}^{7}\left[I(i+k+r, j+l+c) - P(i+r, j+c)\right]^2$$

- Mean Absolute Error :

$$MAE(k,l) = \frac{1}{N^2}\sum_{r=0}^{7}\sum_{c=0}^{7} abs\left[I(i+k+r, j+l+c) - P(i+r, j+c)\right]$$

L'implémentation matérielle de l'opérateur SAE consiste à implanter un module matériel de base pour comparer deux blocs de même taille N×N dans deux images successives 'P' et 'I' (avec N=4 à 16 pour la norme H.264/AC et 8 pour les normes précédentes). Le bloc dans l'image 'P' est fixe tandis que le bloc dans l'image 'I' est mobile; le principe de sélection des blocs est bien détaillé par David [2]. La problématique consiste à minimiser le temps de calcul de l'opération SAE.

5. PARALLELISMES PROPOSES

En adoptant des architectures Muti Instruction Multi-Data (MIMD), les trois opérateurs basés sur plusieurs processeurs exécutent différentes instructions sur différentes données. Ces données arrivent, sous forme d'un signal, à partir d'une mémoire externe ou bien de la mémoire interne [2].

L'utilisation des registres de décalage et de rotation (sur des bits, des pixels et des lignes de pixels) permet la réalisation des architectures matérielles extrêmement efficaces.

Ce principe est bien détaillé dans Messaoudi *et al.*[11], il est fortement conseillé pour des applications qui demandent la lecture parallèle des données (Architectures Multi Data).

5.1 Le produit scalaire et les filtres

Le produit scalaire, donné par la formule suivante $PCF = \sum_{i=1}^{8} C_i.A_i$, peut être schématisé sous la forme de la figure 3.

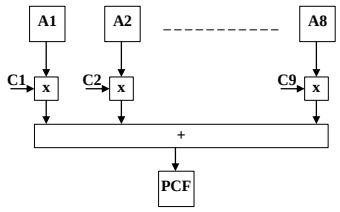

Figure 3 : Architecture parallèle du produit scalaire.

Plusieurs architectures peuvent être supposées pour accélérer l'opération d'addition, avec un calcul du taux de parallélisme pour chaque proposition [8].

La figure 4 montre une première architecture avec un parallélisme égale à : (7 additions + 8 multiplications) × 8 ressources = 120

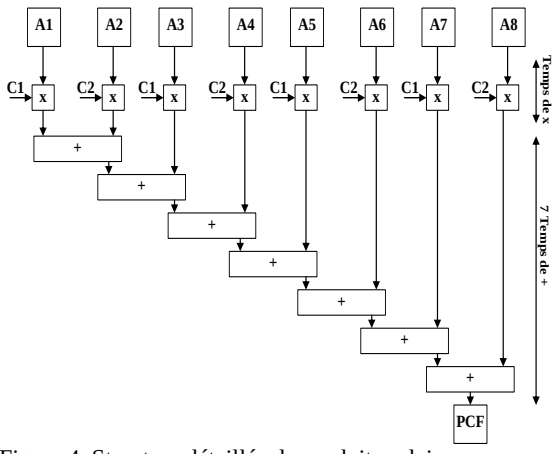

Figure 4. Structure détaillée du produit scalaire

Le taux d'utilisation de chaque opérateur de multiplication est de 100% puisque le calcul complet des sorties est effectué après l'arrivée des données simultanément (architecture multi data MD). Ou bien de 50% si le calcul complet des deux sorties nécessite l'arrivée de deux données (architecture single data SD ou bien adressage d'une mémoire externe). Le taux d'utilisation des opérateurs d'addition n'est pas le même d'une opération à une autre avec une moyenne de 37%.

La figure 5 montre une deuxième architecture pour le produit scalaire. Le temps nécessaire dans une architecture MD est égal au temps de l'opération de multiplication plus 3 temps de l'opération d'addition. Le taux de parallélisme dans cette architecture est égal à (7additions + 8 multiplications) × 8 ressources = 120.

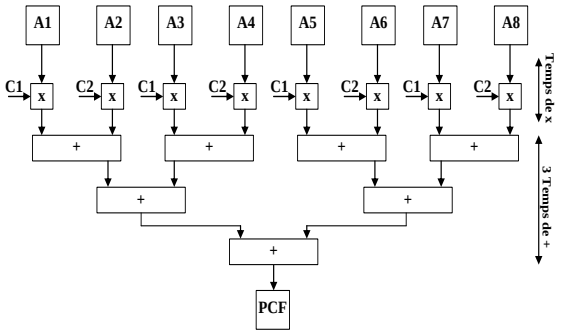

Figure 5. Produit scalaire avec 3 temps d'addition.

Le taux d'utilisation de chaque opérateur de multiplication est de 100% pour une architecture multi data MD, ou bien de 50% pour une architecture single data SD. La moyenne du taux d'utilisation des opérateurs d'addition est de 76%.

Concernant les filtres numériques, nous avons remarqué que la majorité des filtres, et surtout les filtres RIF, utilisent un coefficient de pondération égal à -1, 0 et +1, ce qui rend les calculs plus simples, exemple du filtre de Prewitt [**7**] (Fig.6). Le taux de parallélisme dans cette architecture est égal à (5 additions) × 8 ressources = 40.

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$ Ou bien

$$\mathrm{PCF} = ((\mathrm{A1} - \mathrm{A3}) + (\mathrm{A4} - \mathrm{A6})) + (\mathrm{A7} - \mathrm{A9})$$

L'architecture proposée pour ce filtre est la suivante : le taux d'utilisation des opérateurs d'addition n'est pas le même d'une opération à l'autre avec une moyenne de 76.7 % et le temps

nécessaire pour effectuer l'opération dans une architecture MD est égal au 3 temps d'addition.

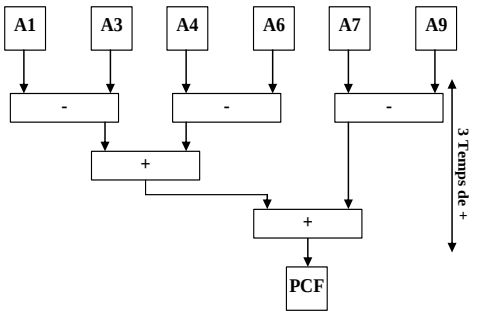

Figure 6. Réalisation du filtre de Prewitt.

5.2 L'opérateur SAE

L'opérateur SAE est un opérateur appliqué sur deux fenêtres de deux images différentes de même taille, ce qui rend l'utilisation d'une architecture MD obligatoire pour assurer le temps réel pour le traitement des images fixes et dynamiques.
Cette architecture nous offre les pixels des deux fenêtres (généralement 8×8 pixels pour chaque fenêtre) en même temps [12].
Le travail d'implantation consiste à minimiser le module de calcul. La soustraction ou bien la somme en complément à 2 des 64 pixels de la première fenêtre avec les 64 pixels de la deuxième fenêtre constitue le premier étage avec 64 additions de ce type (Fig. 7).

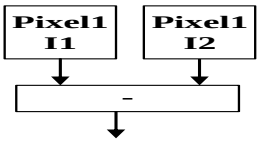

Figure 7. Elément de base dans l'implantation matérielle du SAE.

En sortie du premier étage, on trouve 64 valeurs prêtes pour le calcul de la somme. L'utilisation de 32 additions dans le deuxième étage nous donne 32 valeurs pour le troisième et ainsi de suite (Fig.8). De point de vue programmation, il suffit de programmer un composant (Component) qui simule la soustraction de deux pixels, et d'écrire le programme VHDL (Very high speed integrated circuits Hardware Descripton Language) qui fait appel à ce composant.

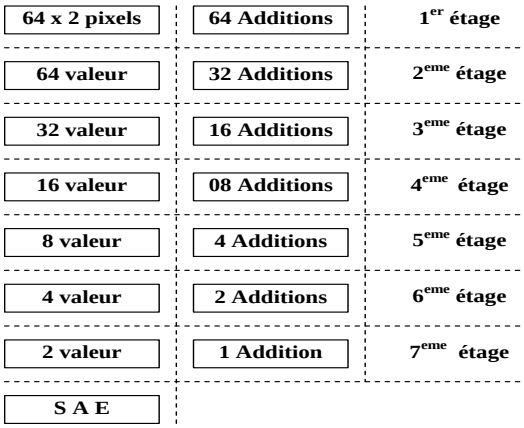

Figure 8. Structure hiérarchique du SAE.

Le taux de parallélisme dans cette architecture est égal à (127 additions) × 64 × 2 ressources = 16256. Le taux d'utilisation des opérateurs d'addition change d'une opération à l'autre avec une moyenne de 50%. Le temps nécessaire pour effectuer l'opération dans une architecture MD est égal aux 7 temps d'addition. Pour minimiser le nombre des additions, on peut à chaque étage utiliser les mêmes composants que l'étage n°1, ce qui rend le nombre maximum des additions utilisées à 32 additions. On peut dans ce cas nommer l'étage comme une étape. Ce qui donne un taux de parallélisme égal à : (32 additions) × 64 × 2 ressources = 4096. Le taux d'utilisation des opérateurs d'addition change d'une opération à une autre avec une moyenne de 50%. Le temps nécessaire pour effectuer l'opération dans une architecture MD est égal aux 7 temps d'addition.
Par contre si on cherche à accélérer les calculs et non pas à minimiser le nombre d'additions, il est intéressant d'utiliser le principe de pipeline qui consiste à utiliser les 7 étages en même temps pour des données différentes. Ce qui donne un taux de parallélisme égal à : (127 additions) × 64 × 2 ressources = 16256.
Le taux d'utilisation des opérateurs d'addition se défère d'une opération à une autre avec une moyenne de 100%. Le temps nécessaire pour effectuer l'opération dans une architecture MD est égal au temps d'addition.
Tous les résultats sont confirmés par la simulation et la synthèse. En effet, les propositions sont écrites en langage VHDL, les simulations sont effectuées avec ModelSim de MentorGraphic, et enfin la synthèse est effectuée avec Xilinx ISE en utilisant plusieurs cartes FPGA de chez Xilinx.

6. L'OPERATEUR SAD DANS H.264

La norme H.264 a été développée pour réaliser des améliorations significatives au-dessus des normes existantes pour la compression des séquences d'images. Cette nouvelle norme visuelle partage un certain nombre de dispositifs avec les normes passées, y compris H.263 et MPEG-4, surtout concernant le module d'estimation et de compensation de mouvement. La seule différence pour ce module consiste en l'utilisation des blocs de pixels de taille 16×16 au lieu de 8×8 dans les normes précédentes.

$$\mathrm{SAE(k,l)} = \sum_{r=0}^{15}\sum_{c=0}^{15} \mathrm{abs}\left[\mathrm{I}(i+k+r,\, j+l+c) - \mathrm{P}(i+r,\, j+c)\right]$$

Deux implantations matérielles de l'opérateur SAE sont possibles : devons-nous utiliser la mémoire interne du circuit FPGA ou bien une mémoire externe ? Dans cette dernière possibilité, nous sommes généralement limités par la taille du bus utilisé. Si nous utilisons le bus Avalon par exemple dans un système sur puce, avec une taille de 32 bits, nous avons la possibilité de lire 4 pixels de 8 bits uniquement à la fois (architecture MultiDada limitée).
Dans le premier cas et selon David [2], l'architecture proposée est purement MultiData où les 16 pixels dans les deux images sont disponibles en même temps pour le bloc de calcul. Ainsi, nous avons la possibilité d'imaginer plusieurs architectures parallèles pour l'opérateur SAE avec un compromis entre le temps de calcul et le nombre des ressources matérielles utilisées (Fig. 9). Des simulations avec ModelSim sur des images réelles sont réalisées pour valider nos propositions.

7. CONCLUSION

Dans ce travail, nous avons étudié et implanté (en matériel) plusieurs opérateurs mathématiques utilisés dans les algorithmes et les normes de compression des images. Le but consiste à accélérer le temps de calcul des opérateurs pour obtenir le temps réel pour des applications irréalisables en utilisant un calculateur séquentiel (PC). Pour cela, nous avons présenté des architectures parallèles pour trois opérateurs (le produit scalaire, le filtrage et l'opérateur SAE) constituant la base des algorithmes, pour les autres opérateurs nous gardons les mêmes idées. Nous avons utilisé le principe de parallélisme des données, ce qui offre la possibilité d'avoir des structures de pipeline, cela va diminuer considérablement le temps de calcul des différents opérateurs, et par conséquent le temps de calcul de l'algorithme total utilisant ces opérateurs. Il nous reste à tester ces architectures pour des algorithmes complets appliqués sur des séquences réelles, et ce, en utilisant des cartes FPGA précises utilisées pour l'implémentation des algorithmes de traitement d'images et de la vidéo.

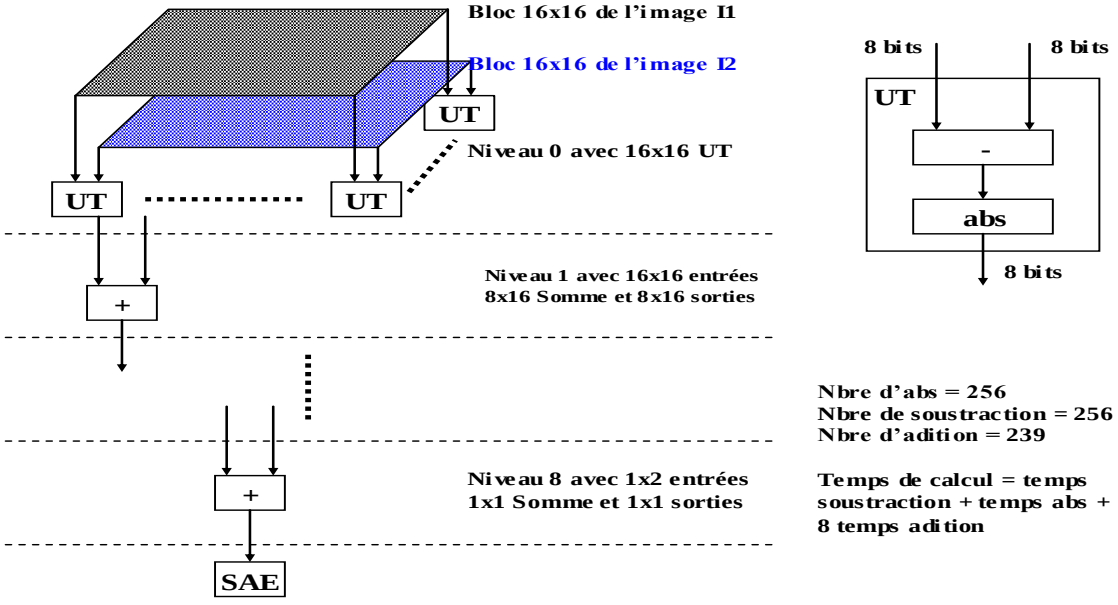

Figure 9. Structure hiérarchique du SAD dans H.264.

REFERENCES

[1] Schäfer R., Wiegand T. & Schwarz H., 2003. H.264/AVC la norme qui monte, UER – *Revue Technique*, Vol. 2003, 1-10.

[2] David J.P., 2002. Architecture synchronisée par les données pour système reconfigurable. Thèse de Doctorat en Sciences, Université catholique de Louvain UCL, 160p.

[3] Ghozzi F. & Nouel P., 2000. Acquisition / Traitement / Restitution – Conception en vidéo temps réel sur composant programmable, JSFT'2000, Tunisie.

[4] Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG, 2003. Draft of Version 4 of H. 264/AVC (ITU-T Recommendation H. 264 and ISO/IEC 14496-10 (MPEG-4 part 10) Advanced Video Coding).

[5] Messaoudi K., Toumi S. & Bourennane E.B., 2008. Material architecture proposition for the block matching method of motion estimate in H.264 standard, IEEE conference, International Conference on Information & Communication Technologies: from Theory to Applications ICTTA'08, Syria.

[6] Ku C. W., Cheng C. C., Yu G. S., Tsai M. C. & Chang T. S., 2006. A High-Definition H.264/AVC Intra-Frame Codec IP for Digital Video and Still Camera Applications, *IEEE Trans*. on Circuits and Systems for Video Tech., Vol. 16 (8), 917-928.

[7] Attitallah A. B., 2007. Etude et Implantation d'Algorithmes de Compression d'Images dans un Environnement Mixte Matériel et Logiciel. Thèse de doctorat en Sciences, Université Bordeaux 1, 240p.

[8] Saidani T., 2008. Traitement d'images parallèle sur les architectures embarquées émergentes, JESTRE, EMP, Algeria.

[9] Messaoudi K., Toumi S. & Bourennane E.B., 2007. Nouvelles architectures matérielles pour la lecture des images statiques et dynamiques en temps réel, JSS'07-Guelma, Algérie.

[10] Site : http://hardh.264.sourceforge.net/ H264-encoder-manual.html.

[11] Messaoudi K., Toumi S. & Bourennane E.B., 2008. Proposal and study for an architecture hardware/software for the implementation of the standard H.264, CISA'08, In: American Institute of Physics (AIP), Vol. 1019 (1), 536-540.

[12] Loukil H., Atitallah A.B., Ghozzi F., Ayed M.A. & Masmoudi N., 2008. A Pipelined FSBM Hardware Architecture for HTDV-H.26x, *International Journal of Electrical Systems Science and Engineering*, Vol. 2008, 536-540.