Conception et validation d'un processeur programmable de traitement du signal à faible consommation et à faible empreinte silicium : application à la vidéo HD sur téléphone mobile

Mathieu Thevenin

▶ **To cite this version:**

Mathieu Thevenin. Conception et validation d'un processeur programmable de traitement du signal à faible consommation et à faible empreinte silicium : application à la vidéo HD sur téléphone mobile. Architectures Matérielles [cs.AR]. Université de Bourgogne, 2009. Français. NNT : . tel-00504704

HAL Id: tel-00504704
https://theses.hal.science/tel-00504704

Submitted on 16 Aug 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Numero d'Ordre : xxxx
EDSPIC : xxx

UNIVERSITÉ DE BOURGOGNE

ÉCOLE DOCTORALE

E2S - ENVIRONNEMENT - SANTÉ/STIC

Formation Doctorale :

Instrumentation et Informatique de l'Image

Thèse

présentée par

Mathieu Thevenin
(Ingénieur-chercheur)

pour obtenir le grade de

DOCTEUR D'UNIVERSITÉ

(SPÉCIALITÉ : Électronique et Informatique)

Conception et validation d'un processeur programmable de traitement du signal à faible consommation et à faible empreinte silicium : application à la vidéo HD sur téléphone mobile

Soutenue en public le 16 octobre 2009 devant le jury :

L. Torres	**Président du jury**
A. Mérigot	**Rapporteur**
O. Sentieys	**Rapporteur**
L. Letellier	**Docteur-ingénieur CEA**
M. Paindavoine	**Directeur de thèse**
B. Heyrman	**Co-directeur de thèse**

Remerciements

Autant de temps aura été nécessaire pour finaliser ces quelques lignes de remerciements que pour la rédaction de l'ensemble de ce mémoire. Malgré ceci, elles ne peuvent exprimer ma gratitude envers les personnes sans lesquelles ces travaux n'auraient pas vus le jour.

Ce travail de thèse a été réalisé au Laboratoire Calculateurs Embarqués et Image (LCEI) du Laboratoire d'Intégration des Systèmes et des Technologies (LIST) au Commissariat à l'Énergie Atomique (CEA) de Saclay dans le Service Architecture et Conception (SARC) du Département et des Techniques des Systèmes Intelligents. La direction académique de ces travaux a été assurée par Michel Paindavoine et Barthélémy Heyrman du Laboratoire d'Electronique d'Informatique et Image (LE2i) de l'Université de Bourgogne.

Je remercie d'abord Denis Platter pour son accueil au sein de son département (DTSI) ainsi que Thierry Collette, responsable du SARC, qui m'a permis de mener mes travaux dans d'aussi bonnes conditions que possibles, tant d'un point de vue technique que logistique. Je tiens à adresser une grande part de mes remerciements à Laurent Letellier qui a su m'offrir un encadrement de qualité au jour le jour, enrichi d'enseignements qui dépassent les activités d'encadrement. La couleur de ces travaux de thèse auraient été sans aucun doute bien différente sans sa rigueur et ses qualités humaines exceptionnelles.

Je tiens à remercier particulièrement Michel Paindavoine, qui, a dirigé cette thèse, contribuant ainsi à la qualité des travaux qui en découlent, et pour sa contribution à mes études secondaires. Il a grandement influé mes orientations professionnelles et ma culture scientifique de par ses enseignements dispensés tout au long des années au cours desquelles j'ai eu la chance d'être un de ses étudiants. Mes remerciements vont à Barthélémy Heyrman qui a codirigé ces travaux avec Michel Paindavoine. Nous avons pu partager des moments privilégiés ainsi que nombre de discussions techniques fort intéressantes, non seulement pendant les trois années qu'aura duré cette thèse, mais aussi pendant mes études secondaires où il aura été un enseignant de qualité.

Je tiens à remercier Lionel Torres qui m'a fait l'honneur de présider mon jury de thèse composé des personnes suscitées, et des rapporteurs. Je les en remercie grandement en saluant leur niveau d'implication et de l'intérêt qu'ils ont portés à ces travaux. Il s'agit d'Alain Mérigot de l'Université de Paris Sud XI et d'Olivier Sentieys de L'ENSSAT de Lannion qui ont rapporté avec beaucoup de rigueur et de précision l'ensemble de recherches à l'issue de ces trois années de thèse.

L'ambiance exceptionnelle du LCEI, devenu LCE, grâce à la bonne humeur de mes collègues et ami(e)s, qui sauront se reconnaître, a rendu le déroulement de ces trois années particulièrement agréables, et m'aura permis de travailler dans des conditions humaines remarquables. L'apport lié à l'expertise technique de ces derniers n'est pas en reste, et je citerais notamment Renaud Schmit qui m'a énormément appris, et ce, bien au delà de sa participation à la validation de l'architecture et à nombre de réflexions qui entourent sa conception. Je souhaite notamment remercier Maria Lecellier qui a su faire preuve d'une grande patience en recherchant les fautes de Français largement disséminées tout au long du manuscrit. Je remercie ma famille et mes amis, notamment ma sœur et mes parents pour qui ma présence ces trois années a été fort limitée.

J'adresse enfin mes remerciements au Dr. Jean-Louis Béal du CHU de Dijon, sans lequel je n'aurais pu réaliser d'études supérieures dans de bonnes conditions ou envisager de poursuivre en thèse. Je tiens à remercier le Dr. Catherine Dormard ainsi que le Pr. Nadine Attal et son équipe de l'hôpital Ambroise Paré (Boulogne Billancourt) qui ont eu un impact décisif sur ma capacité physique à mener à bien ces travaux.

Table des matières

1 Présentation des applications de traitement d'image 7
1.1 L'acquisition des images dans les systèmes embarqués . . . 7
1.1.1 La chaîne d'acquisition d'image . . . 7
1.1.1.1 Le système optique . . . 8
1.1.1.2 Les imageurs . . . 8
1.1.2 Les défauts introduits lors de l'acquisition des images . . . 10
1.1.2.1 Défauts introduits par l'optique . . . 10
1.1.2.2 Les sources de bruits . . . 11
1.2 Présentation des algorithmes de reconstruction d'image . . . 12
1.2.1 Présentation de la chaîne de reconstruction d'image . . . 12
1.2.2 Réduction des bruits et des défauts . . . 13
1.2.2.1 Correction des défauts fixes et du bruit spatial fixe . . . 14
1.2.2.2 Correction du bruit impulsionnel . . . 14
1.2.2.3 Correction du bruit de pixel . . . 14
1.2.2.4 Correction des défauts optiques . . . 15
1.2.3 Correction de la dynamique et des contrastes . . . 15
1.2.3.1 Contrôle d'exposition . . . 16
1.2.3.2 Adaptation de la dynamique et correction des contrastes . . . 16
1.2.3.3 Conclusion sur la correction de dynamique . . . 18
1.2.4 La reconstruction des couleurs . . . 18
1.2.4.1 La correction de la balance des blancs . . . 18
1.2.4.2 Le démosaïquage . . . 19
1.2.5 L'amélioration d'image . . . 21
1.2.5.1 Changement d'espace colorimétrique . . . 21
1.2.5.2 Réduction des artefacts . . . 21
1.2.5.3 Le rehaussement des contours . . . 21
1.2.5.4 Le rehaussement des contrastes et l'amélioration des couleurs . . . 22
1.3 Présentation de chaînes d'amélioration d'image . . . 23
1.3.1 Première chaîne de traitement . . . 23
1.3.2 Seconde chaîne de traitement . . . 24
1.3.3 Troisième chaîne de traitement . . . 25

2 Etat de l'art des processeurs de traitement d'image et vidéo 27
2.1 Présentation des architectures dédiées . . . 27
2.1.1 Les traitements câblés . . . 28
2.1.2 Les architectures à base de coprocesseurs ou d'accélérateurs dédiés . . . 29
2.1.3 Utilisation de composants dédiés dans les SoC . . . 29
2.1.4 Conclusion sur les architectures dédiées . . . 30
2.2 Présentation des architectures reconfigurables . . . 30
2.2.1 Coarse-Grained Reconfigurable Image Stream Processor . . . 31
2.2.2 Multimedia Oriented Reconfigurable Array . . . 32
2.2.3 Reconfigurable Instruction Cell Array . . . 33
2.2.4 Approche DART . . . 33

2.2.5 Autres structures reconfigurables . . . 33
2.2.6 Conclusion sur les architectures reconfigurables . . . 33
2.3 Présentation des architectures programmables . . . 34
2.3.1 Les processeurs spécifiques à une application ASIP . . . 34
2.3.2 Les processeurs de traitement de flux de données . . . 35
2.3.3 Les accélérateurs graphiques Graphics Processing Unit (GPU) . . . 36
2.3.4 Les architectures parallèles Single Instruction Multiple Data (SIMD) . . . 37
2.3.5 Les processeurs de traitement du signal DSP . . . 39
2.3.6 Processeurs généralistes embarqués . . . 40

3 Détermination du modèle architectural 43
3.1 Analyse des algorithmes . . . 43
3.1.1 Méthode d'analyse des algorithmes . . . 44
3.1.1.1 Définition de la méthode d'analyse et d'obtention des résultats . . . 45
3.1.1.2 Définition de la méthode d'analyse du graphe . . . 46
3.1.2 Résultats de l'analyse des algorithmes . . . 47
3.1.2.1 Détermination des besoins en ressources de calcul . . . 47
3.1.2.2 Détermination des données manipulées et de leurs dynamiques . . . 50
3.1.2.3 Recensement des opérations utilisées . . . 53
3.1.2.4 Étude des niveaux de parallélisme . . . 54
3.1.2.5 Détermination du parallélisme spatial . . . 55
3.1.2.6 Détermination du parallélisme de tâches . . . 56
3.1.2.7 Détermination du parallélisme au niveau instructions . . . 56
3.2 Proposition d'un modèle architectural . . . 63
3.2.1 Proposition d'un modèle de processeur de calcul . . . 63
3.2.1.1 Nombre de voies et file de registres . . . 63
3.2.1.2 Sources de données . . . 64
3.2.1.3 Les opérateurs nécessaires . . . 64
3.2.1.4 Look-Up Tables (LUTs) et mémoire de travail . . . 65
3.2.2 Proposition de regroupement des processeurs **SplitWay** . . . 65
3.2.2.1 Mode de fonctionnement . . . 65
3.2.2.2 Budget en surface silicium et consommation électrique . . . 66
3.2.2.3 Détermination des groupes de processeurs . . . 68
3.2.2.4 Communication entre processeurs . . . 69
3.2.2.5 Gestion des données composites . . . 69
3.2.3 Définition du mode d'accès aux données . . . 70
3.2.3.1 Accès au pixel à traiter . . . 71
3.2.3.2 Accès au voisinage d'un pixel . . . 71
3.2.4 Contrôle des processeurs . . . 73
3.2.4.1 Segments de code disponibles . . . 73
3.2.4.2 Détermination du segment de code à exécuter . . . 73
3.2.5 Détermination de l'interconnexion entre les éléments de l'architecture . . . 74
3.2.5.1 Reconfigurabilité gros grain . . . 74
3.2.5.2 Enchaînement des tuiles de calcul . . . 76

4 Conception de l'architecture eISP 79
4.1 Description de l'architecture . . . 79
4.1.1 Description globale de l'architecture . . . 79
4.1.2 Fonctionnement des tuiles de calcul . . . 80
4.1.2.1 Le mode d'accès aux données . . . 81
4.1.2.2 Adaptation de la fréquence de fonctionnement . . . 81
4.1.2.3 Utilisation de processeurs Very Large Instruction Word (VLIW) deux voies . 81
4.1.2.4 Gestion des métadonnées . . . 81
4.1.2.5 L'unité de contrôle . . . 81
4.1.2.6 Le module de communication . . . 82
4.2 Implémentation du gestionnaire de voisinages . . . 82
4.2.1 Mécanisme de mémorisation des données . . . 82
4.2.1.1 Utilisation de mémoires ligne . . . 83
4.2.1.2 Mise en forme des voisinages . . . 83
4.2.2 Performance, surface et consommation du gestionnaire de voisinages . . . 84
4.2.2.1 Débit en lecture . . . 85
4.2.2.2 Surface silicium . . . 85
4.2.2.3 Consommation électrique . . . 85
4.3 Implémentation du processeur de calcul . . . 87
4.3.1 Choix des opérateurs . . . 87
4.3.2 Définition du jeu d'instructions . . . 89
4.3.2.1 Définition des instructions utilisateurs . . . 90
4.3.3 Accès aux données . . . 94
4.3.4 Réalisation du Pipeline . . . 95
4.3.4.1 L'étage FETCH . . . 95
4.3.4.2 L'étage DECODE . . . 96
4.3.4.3 L'étage EXECUTE . . . 96
4.3.5 Gestion de la consommation . . . 96
4.3.5.1 Le mode « sommeil profond » . . . 96
4.3.5.2 Le mode « sommeil » . . . 97
4.3.5.3 Le mode « léthargie » ou « attente » . . . 97
4.3.6 La file de registres . . . 97
4.4 Support des métadonnées . . . 97
4.4.1 Mécanisme d'accès aux métadonnées . . . 98
4.4.1.1 Exemple d'utilisation des métadonnées . . . 98
4.4.1.2 Accès aux métadonnées . . . 98
4.4.1.3 Implications architecturales . . . 98
4.4.2 Implémentation de l'accès aux métadonnées . . . 99
4.4.2.1 Présentation des registres de masques . . . 99
4.4.2.2 Méthode d'accès aux métadonnées en lecture . . . 99
4.4.2.3 Méthode d'accès aux métadonnées en écriture . . . 100
4.4.2.4 Gestion des métadonnées simplifiée . . . 101
4.5 Unité de contrôle . . . 101
4.5.1 Modes de fonctionnement . . . 101
4.5.1.1 Mode de fonctionnement SIMD . . . 102
4.5.1.2 Mode de fonctionnement Multiple Single Instruction Multiple Data (Multi-SIMD) . . . 102

4.5.2 Implémentation de l'unité de contrôle . . . 102
4.5.2.1 Caractéristiques générales . . . 102
4.5.2.2 Implémentation complète . . . 103
4.5.2.3 Implémentation simplifiée . . . 103
4.6 L'unité de communication . . . 104
4.6.1 Dimensionnement du bus . . . 104
4.6.2 Implémentation . . . 105
4.6.2.1 Accès au données du bus . . . 106
4.7 Intégration d'extensions à l'architecture . . . 107
4.7.1 Introduction d'unités fonctionnelles au niveau de la tuile de calcul . . . 108
4.7.2 Introduction d'extensions au niveau du processeur de calcul . . . 108
4.7.2.1 Extension du jeu d'opérateurs . . . 108
4.7.2.2 Ajout de coprocesseurs . . . 108

5 Génération et validation de l'architecture 113
5.1 Estimation des surfaces et de la consommation . . . 114
5.1.1 Paramètres influant sur la surface et la consommation de l'architecture . . . 114
5.1.1.1 Les processeurs *SplitWay* . . . 114
5.1.1.2 Surface et consommation des gestionnaires de voisinages . . . 115
5.1.1.3 Surface et consommation de l'unité de contrôle . . . 115
5.1.1.4 Le module de communication . . . 115
5.1.1.5 Les extensions à l'architecture . . . 115
5.1.2 Génération automatique de l'architecture . . . 116
5.1.3 Obtention des surfaces . . . 116
5.1.3.1 Synthèse ASIC de l'architecture . . . 117
5.1.3.2 Résultats de l'étude en surface post-synthèse . . . 117
5.1.4 Caractérisation des consommations . . . 121
5.2 Caractérisation des performances de l'architecture . . . 126
5.2.1 Estimation de la capacité de calcul totale . . . 126
5.2.2 Capacité effective de calcul . . . 129
5.2.3 Capacité de calcul par unité de surface . . . 130
5.2.4 Capacité de calcul par unité de consommation . . . 131
5.3 Création d'instances de l'architecture adaptée au traitement HD 1080p . . . 132
5.3.1 Programmation de l'architecture eISP . . . 133
5.3.1.1 Particularités de la programmation de l'architecture eISP . . . 133
5.3.1.2 Exemple de programmation d'une tuile de calcul . . . 135
5.3.2 Création d'une instance générique pour le traitement vidéo HD 1080p . . . 137
5.3.2.1 Ressources spécifiques au portage d'algorithmes . . . 137
5.3.2.2 Problématique de la fusion d'algorithmes . . . 140
5.3.2.3 Génération des tuiles de calcul . . . 140
5.3.2.4 La répartition des algorithmes sur les tuiles de calcul . . . 141
5.3.2.5 Résultats obtenus . . . 141
5.3.3 Exemple d'instances d'eISP d'un millimètre carré . . . 142
5.3.3.1 Capacité de calcul privilégiée . . . 143
5.3.3.2 Flexibilité privilégiée . . . 144
5.4 Positionnement par rapport à l'état de l'art . . . 145
Bibliographie . . . 161

Table des figures

1 APPARITION DE NOUVELLES APPLICATIONS DE TRAITEMENT DU SIGNAL EN FONCTION DE L'ÉVOLUTION DES TECHNOLOGIES D'INTÉGRATION. 2

1.1 EXEMPLE DE CHAÎNE D'ACQUISITION D'IMAGE. 8
1.2 DISPOSITION DU FILTRE DE BAYER. 9
1.3 EXEMPLE DE VIGNETTAGE. 10
1.4 EXEMPLE DE BRUIT DE COLONNE. 11
1.5 EXEMPLE DE BRUIT DE PIXEL 11
1.6 SCHÉMA SIMPLIFIÉ DES CLASSES DE TRAITEMENT DE LA CHAÎNE DE RECONSTRUCTION D'IMAGE. 13
1.7 IMAGES PHOTOGRAPHIQUES DU *Mont Blanc* RECONSTRUITE (a) IMAGE BRUTE (b) RECONSTRUITE. 14
1.8 EXEMPLE DE VUES DU PHARE (a) ORIGINALE, (b) (c) PRÉSENTANT UNE MAUVAISE RÉPARTITION DES TONS ET (d) TRAITÉE PAR L'ALGORITHME *Retinex*. 17
1.9 PIXELS MANQUANTS (REPRÉSENTÉS PAR « *?* ») LORSQUE CHAQUE PIXEL DE L'IMAGE BRUTE EST ASSOCIÉ AU PLAN COULEUR LUI CORRESPONDANT. 20
1.10 EXEMPLE DE MOIRÉ. 21
1.11 PREMIÈRE CHAÎNE DE RÉFÉRENCE POUR LA RECONSTRUCTION D'IMAGE. 23
1.12 SECONDE CHAÎNE DE RÉFÉRENCE POUR LA RECONSTRUCTION D'IMAGE. 24
1.13 TROISIÈME CHAÎNE DE RÉFÉRENCE POUR LA RECONSTRUCTION D'IMAGE. 25

2.1 SCHÉMA DU PROCESSEUR (a) TRAITEMENT D'IMAGE *DM310* DE *Texas Instruments (TI)*;(b) ET RECONFIGURABLE *Coarse-Grained Reconfigurable Image Stream Processor (CRISP)*. 31
2.2 ORGANISATION SCHÉMATIQUE DE L'ARCHITECTURE *Multimedia Oriented Reconfigurable Array (MORA)* (a) MATRICE 2×2 DE 16 CELLULES RECONFIGURABLES ET LEURS INTERCONNEXIONS;(b) SCHÉMATISATION D'UNE CELLULE RECONFIGURABLE. 32
2.3 MATRICE DE *Reconfigurable Instruction Cell Array (RICA)*. 33
2.4 ORGANISATION DE L'ARCHITECTURE *FIESTA*. 36
2.5 ORGANISATION DE L'ARCHITECTURE *NVIDIA TESLA*. 37
2.6 REPRÉSENTATION SCHÉMATIQUE DE L'ORGANISATION DE *SIMPil* SUR LE PLAN FOCAL. 38
2.7 ORGANISATION DE L'ARCHITECTURE *Hive'flex* DE *Silicon'Hive* (a) UNE TUILE *VSP* QUI CONTIENT LES TROIS MODULES DMA, *ACE* ET *VCE* ;(b) PLUSIEURS TUILES ENCHAÎNÉES POUR RÉALISER UNE APPLICATION DE TRAITEMENT D'IMAGE. 39
2.8 ARCHITECTURE SCHÉMATISÉE DU DSP *TI C64x*. 40

3.1 MÉTHODE DE GÉNÉRATION DU GRAPHE D'EXÉCUTION ET D'OBTENTION DES RÉSULTATS DE PROFILAGE À PARTIR D'UN ALGORITHME DE *Modular Assembler Simulator (MAsS)*. 44
3.2 TRADUCTION DE LA FONCTION $y = a \times x + b$ EN OPÉRATIONS ÉLÉMENTAIRES PUIS EN GRAPHE D'EXÉCUTION. 46
3.3 CAPACITÉS DE CALCUL NÉCESSAIRES À L'EXÉCUTION DES CHAÎNES D'AMÉLIORATION D'IMAGE DE RÉFÉRENCE INTRODUITES DANS LE PREMIER CHAPITRE. 49
3.4 RÉPARTITION DES DIFFÉRENTES OPÉRATIONS EN FONCTION DE LEURS « RÔLES » RESPECTIFS. . . 54
3.5 MISE EN ÉVIDENCE DU PARALLÉLISME SPATIAL SUR LE GRAPHE D'EXÉCUTION D'UN SEUILLAGE, ON VOIT QU'IL EST POSSIBLE D'EXÉCUTER LES TRAITEMENTS SUR CHAQUE PIXEL SUCCESSIF INDÉPENDAMMENT LES UNS DES AUTRES. 55

3.6 TAUX DE REMPLISSAGE MOYEN DES VOIES DE CALCUL EN FONCTION DE LEUR NOMBRE – MÉTHODE PAR ANALYSE STATIQUE DES CODES APPLICATIFS, OPÉRATEURS DE REGISTRE À REGISTRE. 58
3.7 ALGORITHME D'ESTIMATION DU NIVEAU DU PARALLÉLISME MOYEN AU NIVEAU INSTRUCTION POUR UN GRAPHE D'EXÉCUTION DONNÉ. 59
3.8 MÉTHODE EMPLOYÉE POUR L'EXTRACTION DU PARALLÉLISME AU NIVEAU INSTRUCTIONS. . . . 60
3.9 TAUX DE REMPLISSAGE MOYEN DES VOIES DE CALCUL EN FONCTION DE LEUR NOMBRE – MÉTHODE PAR ANALYSE DES GRAPHES D'EXÉCUTION DES APPLICATIONS, OPÉRATEURS DE REGISTRE À REGISTRE. 61
3.10 TAUX DE REMPLISSAGE MOYEN DES VOIES DE CALCUL EN FONCTION DE LEUR NOMBRE – MÉTHODE PAS ANALYSE DES GRAPHES D'EXÉCUTION DES APPLICATIONS, OPÉRATEURS À ACCÈS DIRECT. 61
3.11 A GAUCHE :PROGRAMME UTILISANT DES OPÉRATEURS REGISTRE À REGISTRE, À DROITE, LE MÊME PROGRAMME UTILISANT DES OPÉRATEURS À ACCÈS DIRECT. 62
3.12 EXEMPLE DE CODE TRAITANT DES IMAGES BRUTES, DÉCRIT AVEC DES BLOCS CONDITIONNELS À GAUCHE, ET DES OPÉRATIONS CONDITIONNELLES DE TYPE SIMD À DROITE. 66
3.13 EXEMPLE SCHÉMATIQUE DE L'EXÉCUTION D'OPÉRATIONS TRAITANT DES IMAGES BRUTES SUR UNE STRUCTURE SIMD. 67
3.14 EXEMPLE SCHÉMATIQUE DE L'EXÉCUTION D'OPÉRATIONS TRAITANT DES IMAGES BRUTES SUR UNE STRUCTURE MULTI-SIMD. 67
3.15 EXEMPLE D'ENCHAÎNEMENT DE PLUSIEURS TRAITEMENTS SUR TUILES DE CALCUL REGROUPANT DES PROCESSEURS EN MODE MULTI-SIMD. 68
3.16 EXEMPLE DE LECTURE D'UNE MÉTADONNÉE DE 10 BITS DANS UN MOT DE 24 BITS QUI CONTIENT 3 VALEURS. SA VALEUR EST AUTOMATIQUEMENT EXTRAITE À L'AIDE DU MASQUE DE LECTURE CONFIGURÉ AU PORTAGE DU TRAITEMENT 70
3.17 SCHÉMA ET CHRONOGRAMME DU PROCESSUS DE DISTRIBUTION DES PIXELS AUX DIFFÉRENTS PROCESSEURS D'UNE MÊME TUILE DE CALCUL. (A) SCHÉMA DE PRINCIPE DE LA PARALLÉLISATION DES PIXELS VERS LES Nb_{Procs} PROCESSEURS DE CALCUL (B) CHRONOGRAMME DES SIGNAUX CORRESPONDANT AUX FLUX PARALLÉLISÉS POUR $Nb_{Procs} = 4$ PROCESSEURS DE CALCUL, ON PEUT VOIR QUE LES PIXELS SONT DISPONIBLES PENDANT Nb_{Procs} PÉRIODES PIXELS. 71
3.18 HIÉRARCHIE MÉMOIRE PERMETTANT LA DISTRIBUTION DES VALEURS DES PIXELS ET DE LEURS VOISINAGES AUX PROCESSEURS D'UNE TUILE DE CALCUL. 72

4.1 EXEMPLE DE TUILES INTERCONNECTÉES ENTRE ELLES PAR LE BUS TIME DIVISION MULTIPLE ACCESS (TDMA). 80
4.2 EXEMPLE DE DISTRIBUTION D'UN FLUX DE DONNÉES SUR 4 PROCESSEURS. 83
4.3 ALGORITHMES UTILISÉS PAR LES MACHINES À ÉTAT DES GESTIONNAIRES DE VOISINAGES (A) POUR LA MISE À JOUR DES MÉMOIRES LIGNE, (B) POUR LA MISE À JOUR DES REGISTRES DE TAMPONS. 84
4.4 EXEMPLE DE CHRONOGRAMME DE L'ACCÈS AUX VOISINAGES POUR UNE TUILE DE 4 PROCESSEURS. 84
4.5 SÉQUENCE DE RECONSTITUTION DES VOISINAGES POUR 4 PROCESSEURS À PARTIR DU FLUX DE PIXELS (DE HAUT EN BAS). 86
4.6 RÉORGANISATION DES LIGNES LORS DE LA RECONSTRUCTION DES VOISINAGES POUR 4 PROCESSEURS. 86
4.7 LES REGISTRES DE 4 VOISINAGES 5×5 MUTUALISÉS POUR 4 PROCESSEURS. 87
4.8 EXEMPLE D'UN ALGORITHME DE BINARISATION, LA PARTIE `IF` EST RÉALISÉE SUR LA PREMIÈRE VOIE, TANDIS QUE LA PARTIE `ELSE` EST RÉALISÉE SUR LA DEUXIÈME VOIE. LA MÉTADONNÉE 2 PERMET D'ENREGISTRER LA VALEUR DU BIT CONJOINTEMENT À LA VALEUR DU PIXEL. 94
4.9 SCHÉMA SYNOPTIQUE DU PROCESSEUR *SplitWay*. 95

4.10 Exemples de quelques registres de masques utilisés pour les métadonnées. De gauche à droite, les masques tels qu'ils sont décrits par le programmeur, les masques tels qu'ils sont implémentés, et le résultat de l'extraction d'une métadonnée avec chacun de ces masques. 100
4.11 Logique permettant l'accès aux métadonnées `i` associées par les registres de masques `Mi` dans les cas (a) d'accès en lecture et (b) d'accès en écriture.Les nombres signés sont supportés en propageant la valeur de bit du signe lors du décalage des valeurs. 100
4.12 Connexion de l'unité de contrôle Multi-SIMD avec les processeurs *SplitWay* dans les cas où (a) le mode Multi-SIMD complet commandé par les métadonnées ou le type de pixel brut traité, (b) le Multi-SIMD simplifié est adapté au traitement d'image brutes. 103
4.13 Détermination graphique du nombre de slots du bus TDMA en fonction des fréquences des tuiles de calcul F_1 à F_3. 105
4.14 Exemple de chronogramme pour un bus TDMA utilisant deux canaux de communication. 105
4.15 Exemple de stratégies de connexion des tuiles de calcul. 106
4.16 Module d'entrée sortie permettant de lire et écrire sur le bus TDMA. 106
4.17 Exemple schématisant l'ordonnancement des accès au niveau du bus TDMA, à gauche la configuration d'enchaînement voulue, à droite l'activité des composants sur chaque slot disponible pendant une période pixel. 107
4.18 Exemple d'intégrations de composants dédiés au sein des tuiles de calcul, (a) UnitÃ©s fonctionnelles (UFs) intégrées au flux de données, (b) unité fonctionnelle accessible en tant qu'opérateur au sein de l'unité Execute du processeur, (c) coprocesseur alimenté en données par le gestionnaire de voisinages et/ou par le processeur via l'espace d'adressage du plan mémoire. 109
4.19 Représentation de l'architecture dans son ensemble, plusieurs tuiles de calcul qui peuvent être hétérogènes sont interconnectées entre-elles. 110

5.1 Répartition de la surface entre les différents éléments du processeur *SplitWay* 24 bits. 118
5.2 Surface dédiée aux processeurs *SplitWay* et surface dédiée au gestionnaire de voisinages en fonction du nombre de processeurs pour une tuile de calcul donnée. Le nombre de processeurs à partir duquel la surface de calcul devient plus importante que la surface des éléments de mémorisation (en fonction de la taille des voisinages gérés) sont indiqués – Processeurs *SplitWay* 24 bits et mots de données de 8 bits, surface normalisée par rapport à une tuile de calcul de deux processeurs sans gestionnaire de voisinages. 119
5.3 Évolution de la surface normalisée de l'architecture en fonction de la taille de mots à traiter et de la taille du voisinage $N \times M$. 120
5.4 Évolution de la surface d'une tuile de calcul en fonction du nombre de processeurs *SplitWay* et de la taille des voisinages $N \times M$ gérés – Processeurs *SplitWay* 24 bits intégrant 256 mots de 24 bits de mémoire de travail et capable de traiter des mots de 24 bits (soit 3 pixels de 8 bits). 120
5.5 Résultat du placement routage d'une tuile de 6 processeurs *SplitWay* 24 bits, chacun associé à une mémoire de travail 256 mots de 24 bits et d'un gestionnaire de voisinage 3×3, la surface totale obtenue 0,26 mm^2 pour 272 MHz en technologie 65 nm. . 122

5.6 Évolution de la consommation électrique en fonction du nombre de processeurs – Processeurs *SplitWay* 24 bits 125
5.7 Évolution de la consommation électrique normalisée en fonction de la taille des voisinages. 125
5.8 Capacité de calcul à 250 MHz par unité de surface en exprimée GOPs/mm^2 en fonction du nombre de processeurs et de la taille des voisinages utilisés – Processeurs *SplitWay* 24 bits associés à des mémoires de travail de 256 mots de 24 bits. 131
5.9 Capacité de calcul à 200 MHz par unité de surface en exprimée MOPs/mW en fonction du nombre de processeurs et de la taille des voisinages utilisés – Processeur *SplitWay* 24 bits associé à des mémoires de travail de 256 mots de 24 bits. 132
5.10 Exemple de portage de prédicat `IF` à l'aide d'instructions conditionnelles, ici le drapeau utilisé est le drapeau `F0`, les préfixes `F0` et `NF0p` permettent respectivement d'exécuter l'opération sur la valeur à vrai et à faux du drapeau. 134
5.11 Cœur de boucle de l'algorithme de démosaïquage bilinéaire. Cette implémentation fait apparaître les différentes équations à appliquer en fonction de la couleur du pixel à traiter. 136
5.12 Le programme de l'algorithme de démosaïquage bilinéaire tel qu'il est porté sur une tuile de l'architecture *eISP*. 138
5.13 Portage de la première chaîne de référence sur *Vid'eISP.1*, le taux d'utilisation de la tuile de calcul est présenté pour la fréquence de fonctionnement choisie, et par rapport à sa capacité de calcul à 250 MHz, deux tuiles sont inutilisées, (*WB* signifie « balance des blancs »). 141
5.14 Portage de la seconde chaîne de référence sur *Vid'eISP.1*, le taux d'utilisation de la tuile de calcul est présenté pour la fréquence de fonctionnement choisie, et par rapport à sa capacité de calcul à 250 MHz, toutes les tuiles sont utilisées, mais une importante capacité de calcul reste disponible (*WB* signifie « balance des blancs »). 142
5.15 Portage de la troisième chaîne de référence sur *Vid'eISP.1*, le taux d'utilisation de la tuile de calcul est présenté pour la fréquence de fo nctionnement choisie, et par rapport à sa capacité de calcul à 250 MHz, deux tuiles sont inutilisées (*WB* signifie « balance des blancs »). 142
5.16 Latence du bus TDMA implémenté en anneau, en mode point à point (trait plein) et en mode pleinement connecté (trait en pointillé). 155

Liste des tableaux

3.1 Ressources de calcul requises pour chaque algorithme, exprimées en opérations par pixel et en GOPs pour des flux vidéo Haute Définition (HD). 48
3.2 Répartition des ressources de calcul en fonction des rôles assignés aux opérations. 50
3.3 Ressources de calcul requises pour la partie calcul de chaque algorithme, exprimées en opérations par pixel et en GOPs pour des flux vidéo HD. 51
3.4 Taille des masques de voisinages typiquement utilisés, et format des images d'entrée et de sortie des algorithmes étudiés. `B` pour image brute, `C1` pour un seul plan couleur ou en niveaux de gris et `C3` pour les images comportant trois plans (ex *RVB*, *YUV* etc.) 52
3.5 Résumé des choix architecturaux possibles pour le processeur élémentaire. 62

4.1 Répartition moyenne des opérations pour les algorithmes usuels de traitement d'image lorsqu'un jeu d'instructions orthogonal est utilisé. 88

5.1 Surface des différents éléments du processeur *SplitWay* en fonction de la taille du chemin de données et opérateurs (ici 16 et 24 bits) en technologie TSMC 65 nm. . . . 118
5.2 Résultats de l'étude en consommation pour une tuile de calcul comportant 6 processeurs *SplitWay* de 24 bits traitant des mots de données 8 bits en HD 1080p. 124
5.3 Nombre d'opérations disponibles par pixel en fonction du nombre de processeurs et de la fréquence pour un flux vidéo HD 720p à 25 images par seconde. 129
5.4 Nombre d'opérations disponibles par pixel fonction du nombre de processeurs de et de la fréquence pour un flux vidéo HD 1080p à 25 images par seconde. 130
5.5 Taux d'utilisation des processeurs en fonction de leur fréquence de fonctionnement. 137
5.6 Propriétés des algorithmes de la première chaîne de référence portés sur l'architecture *eISP*. 139
5.7 Propriétés des algorithmes de la seconde chaîne de référence portés sur l'architecture *eISP*. 139
5.8 Instance semi-hétérogène d'eISP, capable de délivrer 40 GOPs à 250 MHz pour 295 mW de consommation électrique sur 2,17 mm^2 de surface silicium 141
5.9 Tableau de synthèse présentant les tuiles sélectionnées pour une instance de l'architecture *eISP* privilégiant la capacité de calcul. 143
5.10 Tableau de synthèse présentant les tuiles sélectionnées pour une instance de l'architecture *eISP* privilégiant la flexibilité. 144
5.11 Comparaison de différentes instances d'*eISP* avec différentes architectures de l'état de l'art. 147
5.12 Comparaison de différentes instances d'*eISP* avec les cinq architectures représentatives de l'état de l'art. 148

Introduction

Le contexte de cette étude

AUJOURD'HUI, plus de capteurs d'images numériques ont été vendus dans le monde que d'appareils photographiques argentiques et cameras vidéo l'ont été depuis l'invention de la photographie. Le marché dépasse ainsi les 8 milliards [Tufegdzic 2009] de dollars et il ne cesse de croître. En effet, chaque jour apporte son lot de nouvelles applications et les coûts de fabrication diminuant, la diffusion de ces produits devient très forte. Le marché des capteurs est fortement lié à celui de la téléphonie mobile puisque les analystes attendent pour l'année 2010 la vente de 1,2 milliard de capteurs exclusivement pour la téléphonie mobile, contre seulement 350 millions pour d'autres dispositifs.

Les imageurs sont maintenant industrialisés sous forme de modules qui intègrent un système optique, un capteur et une électronique de traitement dans un ensemble dont le prix de revient n'excède pas quelques dollars [Global Sources 2009], moins d'un dollar d'ici la fin de l'année. Pour arriver à de tels coûts, les éléments liés à l'optique sont simplifiés au maximum, et la surface silicium, qui est le premier facteur de coût des composants électroniques, est réduite à l'extrême. Aujourd'hui, les capteurs ont une taille de 1/10 de pouce, et la taille des modules complets ne dépasse pas quelques millimètres de côté. Cette miniaturisation induit inévitablement des images de faible qualité, la surface dédiée à l'électronique d'amélioration de l'image ne dépasse généralement pas quelques millimètres carrés en utilisant les technologies actuelles d'intégration (65 nm basse consommation). Ainsi la surface dédiée aux photosites est considérablement réduite (actuellement entre 2,2 μm et 1,1 μm de côté alors que la longueur d'onde de la lumière visible commence à 0,8 μm environ) pour permettre la diminution de la surface silicium utilisée tout en augmentant la résolution. De leurs côtés, les consommateurs souhaitent obtenir des images toujours plus résolues, comparables à celles des capteurs de taille 10 à 100 fois plus importantes, c'est à dire de plusieurs millions de pixels (MPixels). A coût égal, c'est donc au niveau des traitements électroniques associés au capteur que se situe une marge de progression vers une meilleure qualité de l'image. C'est pourquoi la reconstruction des images à la sortie des capteurs est primordiale, sans quoi les images ne seraient pas directement utilisables par la consommateurs.

Tout en réduisant les coûts de production, les intégrateurs doivent proposer des nouvelles fonctionnalités qui exploitent ces images haute résolution afin d'étendre leur parts de marché. La Figure 1 illustre brièvement quelques unes des applications développées durant ces dernières décennies et déclinées en téléphonie mobile. On voit que le traitement du signal a une place prépondérante et que la vidéo numérique est une application qui émerge depuis quelques années. La multiplication des applications entraîne la multiplication de standards qu'un même dispositif doit être en mesure de supporter, mais aussi l'augmentation des ressources matérielles nécessaires à leur exécution. Les géants du marché proposent sans cesse de nouvelles applications basées sur l'image.

Aujourd'hui, si les téléphones portables proposent une fonctionnalité de prise de vue et de vidéo, leur résolution reste limitée aux formats Video Graphic Array (VGA) ou D-1 (respectivement 640×480 pixels et 720×576 pixels de résolution à 25 ou 30 images par seconde), alors que le format vidéo actuel, la Haute Définition (HD) 1080p (1920×1080 pixels avec un débit au mojns supérieur à 25 images par seconde) s'impose dans les dispositifs mobiles. Traiter des vidéos HD 1080p en temps réel revient à traiter près de 52 MPixels par seconde. Ceci, alors que le domaine de la téléphonie mobile est sans doute l'un des plus contraints. La consommation

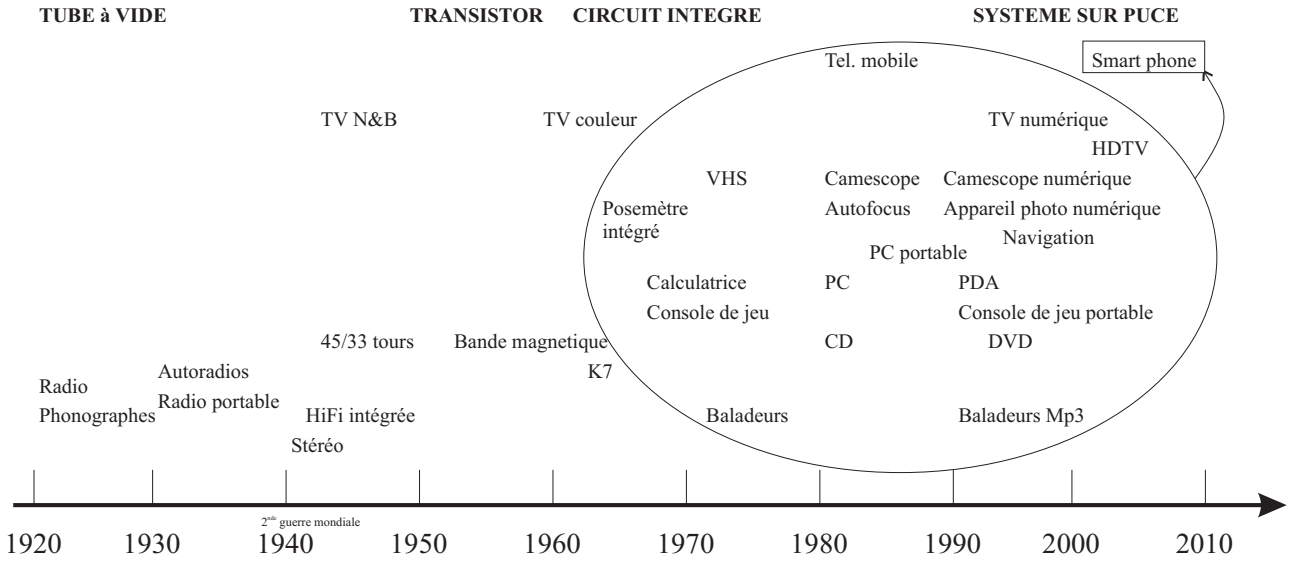


FIG. 1: APPARITION DE NOUVELLES APPLICATIONS DE TRAITEMENT DU SIGNAL EN FONCTION DE L'ÉVOLUTION DES TECHNOLOGIES D'INTÉGRATION.

électrique consentie est particulièrement faible, un téléphone mobile ne consomme pas plus d'un Watt lorsqu'il effectue des opérations gourmandes en énergie [Mayo 2004] comme la transmission de données, et il ne consomme que quelques dizaines de milliwatts lorsqu'il est en veille. La consommation électrique consentie aux dispositifs d'amélioration de l'image issue du capteur est de quelques centaines de milliwatts alors que le volume de données à traiter et les ressources de calcul nécessaires augmentent avec la résolution des imageurs.

Si les imageurs sont de qualité comparable d'un fabricant à l'autre, les choix réalisés en termes d'algorithmes de reconstruction et de traitement qui leur sont associés sont déterminants pour la qualité des images. Chaque fabricant cherche donc à se démarquer de la concurrence en intervenant sur la chaîne algorithmique, avec bien souvent des algorithmes propriétaires. Les approches de reconstruction, de correction et d'amélioration d'image en aval du capteur sont particulièrement variées [Ramanath 2002] et la conception algorithmique d'une chaîne de reconstruction d'image [Kao 2006] est un compromis entre la qualité des images obtenues et le respect des contraintes de surface et de consommation. Dans ce contexte, il est stratégique pour les intégrateurs de disposer d'un calculateur embarqué suffisamment flexible pour supporter un large éventail d'algorithmes répondant aux contraintes de surface silicium et de consommation électrique.

Les travaux actuels

Aujourd'hui, l'amélioration d'image en temps réel au sein des dispositifs mobiles est le plus souvent réalisée par des composants dédiés. Ceux-ci permettent de répondre efficacement aux contraintes de surface et de consommation avec un coût de fabrication très faible. Toutefois, si ces composants s'avèrent performants, ils ne permettent pas de supporter la diversité des algorithmes proposés. En effet, ce type de composant fige les traitements dans le silicium dès sa phase de conception, ce qui rend leur modification impossible sans la création d'un nouveau composant. Des circuits reconfigurables « gros grain » sont proposés pour adapter le circuit en fonction du traitement à exécuter, dans la limite des traitements définis au départ.

Des méthodes sont proposées pour la génération automatique de processeurs d'applications (Application Specific Processor (ASIP)) [Cousin 2000, Jain 2001a]. Les composants ainsi générés se révèlent performants et flexibles pourvu que les applications à supporter aient été initialement intégrées au champ d'étude. La surface du composant va augmenter avec sa flexibilité, le concepteur doit donc rechercher un compromis entre les performances qu'il veut atteindre et les contraintes qui lui sont imposées. Ces composants sont aujourd'hui largement utilisés au sein de dispositifs embarqués.

Les processeurs usuels de traitement du signal, tout comme les processeurs généralistes, ne permettent pas d'atteindre les capacités de calcul requises pour les traitements visés mais offrent la flexibilité permettant l'exécution d'une grande diversité de traitements. Des approches à base d'extensions accélératrices et de co-processeurs permettent de les exploiter efficacement. Ces extensions restent toutefois dédiées à un tâche définie dès la conception du processeur. Le choix des extensions doit donc faire l'objet d'études spécifiques.

Les approches actuelles permettent de répondre aux contraintes de surface silicium et de consommation électrique, mais la flexibilité requise pour supporter la grande variété de traitements n'est pas au rendez-vous. En effet, les circuits sont conçus pour une gamme de traitements donnés, et ainsi lorsqu'un nouvel algorithme doit être utilisé, ou lorsque le produit est fortement modifié, le circuit doit entièrement être revu ou remplacé. Les temps de mise sur le marché sont donc importants, et les produits existants ne peuvent s'adapter aux nouvelles applications ou standards.

L'objectif de cette étude

Comme la grande majorité des capteurs est destinée au domaine de l'embarqué et notamment la téléphonie mobile, et que de nombreuses nouvelles applications requérant d'importantes ressources de calcul sont à venir comme la reconstruction et l'amélioration en temps réel de vidéo HD, nous concentrons notre étude à ce domaine. De plus, la surface silicium et la consommation électrique fortement limitées – quelques centaines de milliwatts pour quelques millimètres carrés en technologie 65 nm pour le traitement d'image – ne permettent pas l'utilisation de composants programmables usuels qui apportent la flexibilité recherchée par la grande variété d'algorithmes potentiellement utilisables. Enfin, nous considérons qu'une approche de traitement du signal adaptée aux contraintes de la téléphonie mobile peut être étendue à d'autres domaines d'applications moins contraints.

L'étude proposée dans ce manuscrit consiste à concevoir et valider une architecture de calcul capable de répondre aux contraintes fortes de surface, de consommation et temps réel du domaine de l'embarqué et de la téléphonie mobile particulièrement. Le cahier des charges fixe la surface silicium à deux à trois millimètres carrés et la consommation électrique à un demi watt avec les technologies d'intégration actuelles (65 nm). La capacité de calcul disponible doit être suffisante pour réaliser la reconstruction et l'amélioration d'une vidéo HD 1080p en temps réel tandis que la flexibilité doit permettre l'utilisation de différents types d'algorithmes de traitement d'image.

La démarche employée pour réaliser cette étude

Tout au long de cette étude nous nous attacherons à proposer une architecture de calcul flexible capable de supporter la grande variété d'algorithmes de traitement du signal. Comme les contraintes en surface et en consommation sont fortes, nous avons principalement choisi d'intervenir conjointement sur les différents termes qui déterminent la puissance dynamique consommée par des circuits en technologie Complementary

Metal Oxide Semi-Conductor (CMOS) qui peut être approximée par l'équation 1 [Piguet 2004]. Le terme A correspond au taux d'activité des transistors, C est la capacité globale du circuit, liée au nombre de transistors et d'interconnexions, F la fréquence de fonctionnement, et V la tension de fonctionnement.

$$P = A \times C \times V^2 \times F \tag{1}$$

La limitation du nombre de transistors, par différentes stratégies détaillées tout au long de ce manuscrit, a un impact direct sur la surface silicium et sur la consommation électrique (termes A et C). Nous avons aussi choisi de limiter leur activité (terme A) tout en réutilisant au mieux les ressources mises en place pour ne pas les dupliquer (terme C). Enfin, une approche permettant une sélection de la fréquence de fonctionnement optimale des différents composants de l'architecture réduit la consommation électrique (terme F et potentiellement la tension[1] V).

Ce manuscrit se décompose en cinq chapitres, auxquels il faut ajouter cette introduction, la conclusion suivie des perspectives présentant les voies ouvertes à l'issue des travaux de cette thèse, et enfin des contributions scientifiques apportées par cette thèse.

- Le chapitre 1 présente brièvement les éléments permettant l'acquisition des images ainsi que quelques unes des dégradations qui touchent les imageurs bas coûts. Les algorithmes de reconstruction et de traitement d'image actuellement utilisés dans les systèmes embarqués sont introduits ainsi que quelques approches potentiellement exploitables dans ce contexte, et qui permettent d'appréhender la variété algorithmique disponible. A partir de ces algorithmes, nous proposons ensuite quelques exemples de chaînes de reconstruction et d'amélioration d'image qui seront utilisées par la suite comme référence pour dimensionner et pour valider notre architecture.

- Le chapitre 2 présente un état de l'art des différentes approches architecturales qui peuvent être employées en traitement d'image embarqué, en partant des composants dédiés, puis en augmentant progressivement le degré de flexibilité. Ainsi, les architectures reconfigurables, puis certains Application Specific Processors (ASIPs) sont présentés avant d'étudier les solutions programmables. Cet état de l'art nous permet de retenir ou rejeter certaines approches en fonction de leur capacité à répondre au cahier des charges précédemment fixé.

- Le chapitre 3 consiste à vérifier si les technologies actuelles permettent d'intégrer la capacité de calcul dans les contraintes de surface et de consommation de l'embarqué. Pour cela, nous nous basons sur les chaînes de reconstruction et d'amélioration d'image présentées dans le premier chapitre. Un outil de profilage de l'exécution des algorithmes a été développé à cet effet et est présenté. Les algorithmes qui composent la chaîne de traitement sont ainsi analysés au niveau opération afin d'extraire les différents niveaux de parallélisme ainsi que les opérateurs nécessaires aux calculs. Cette analyse nous conduit à proposer un modèle architectural extensible qui exploite les résultats de cette étude. Les approches de ce modèle sont validées fonctionnellement par une implémentation en *SystemC* des différents éléments qui le composent.

- Le chapitre 4 présente l'implémentation de l'architecture dont le modèle a été décrit dans le chapitre 3. Les choix réalisés pour limiter la surface silicium et la consommation de l'architecture sont détaillés module

1. Ceci est lié au délai de propagation des transistors qui diminue avec la tension ce qui permet potentiellement d'envisager l'utilisation de techniques de variation de la tension d'alimentation.

par module. Cette implémentation est réalisée en VHSIC (VHDL) synthétisable pour Application Specific Integrated Circuit (ASIC) et a pour but d'obtenir une version complète de l'architecture que nous proposons.

- Le chapitre 5 vérifie la viabilité de l'architecture proposée et met en évidence les différents paramètres permettant d'intervenir sur la surface et la consommation. Pour cela un générateur automatique de l'architecture est mis en œuvre afin de produire différentes versions en fonction de critères fixés par le concepteur. Ce chapitre démontre la flexibilité et l'extensibilité du modèle proposé, et montre qu'il répond aux contraintes de surface silicium et de consommation électrique en proposant différentes instances. Les chaînes de traitements présentées dans le premier chapitre sont alors portées sur l'architecture.

Après avoir conclu sur l'ensemble des travaux réalisés dans le cadre de cette thèse, quelques perspectives ouvertes par l'architecture proposée sont évoquées. Certaines d'entre-elles ont d'ores et déjà fait l'objet d'études, elles sont alors précisées.

CHAPITRE 1

Présentation des applications de traitement d'image

Introduction

LA GRANDE majorité des imageurs vendus dans le monde sont destinés à être intégrés à des téléphones mobiles. Ce marché, fortement concurrentiel de la téléphonie mobile, est d'abord dirigé par les coûts de production et par la demande des consommateurs. Ils recherchent des produits capables d'assurer un maximum de fonctionnalités et une qualité d'image supérieure, pour une coût le plus faible possible. En ce qui concerne l'image, ces fonctionnalités vont de la prise de vues d'images photographiques et leur retouche au montage vidéo, la définition des images étant un critère discriminant.

Le coût de fabrication est directement lié à la taille des imageurs, et donc à leur surface silicium, mais aussi au procédé de fabrication qui peut nécessiter plus ou moins d'interventions. Comme la résolution des capteurs tend à augmenter alors que leur surface reste fixe ou tend à diminuer, la surface photosensible devient de moins en moins importante [Chen 2000], tandis que l'utilisation en mode vidéo nécessite des vitesses de lecture de plus en plus élevées. Dans ce contexte, les images issues des capteurs doivent faire l'objet de reconstruction et d'améliorations avant d'être exploitées.

La première section de ce chapitre décrit brièvement quelques unes des déformations causées aux images issues des modules utilisés dans les systèmes embarqués. Les traitements d'amélioration d'image font l'objet de la seconde partie de ce chapitre qui aborde quelques techniques algorithmiques utilisées pour la reconstruction et l'amélioration d'images à la sortie du capteur. Enfin, la troisième section présente trois chaînes de reconstruction et d'amélioration d'image que nous proposons de décrire au fil de ce document.

1.1 L'acquisition des images dans les systèmes embarqués

Cette section présente la chaîne d'acquisition et de reconstruction usuelle d'image. La littérature présente de nombreux travaux sur la capture d'image, que les procédés soient numériques ou non, la méthode consiste à utiliser un système optique pour focaliser une scène sur une surface photosensible. Celle-ci peut être un capteur, un film etc. Dans la suite de ce document, nous nous plaçons dans le cas d'une matrice de pixels photosensible, ce qui représente la très grande majorité des systèmes de capture d'image numérique embarqués d'aujourd'hui, aussi bien dans les téléphones portables, les appareils photos reflex que les camescopes. Dans un premier temps, nous présentons la chaîne d'acquisition telle qu'elle est généralement implémentée. Dans un second temps, sont présentées quelques unes des dégradations introduites par l'optique, puis par le système de capture électronique.

1.1.1 La chaîne d'acquisition d'image

La chaîne d'acquisition d'image est généralement constituée d'un système optique qui projette l'image de la scène sur une surface photosensible comme le schématise la figure 1.1. Une matrice de photodiodes

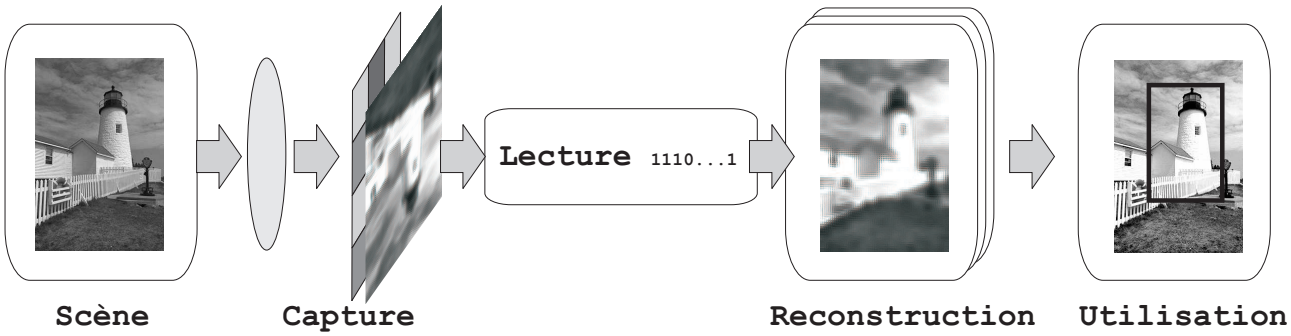


FIG. 1.1: EXEMPLE DE CHAÎNE D'ACQUISITION D'IMAGE.

convertit les photons en électrons, elle constitue la valeur des pixels. Ensuite, ce signal est lu par un dispositif électronique qui le traite afin de reconstituer une image exploitable. Deux parties distinctes du module de la caméra permettent l'acquisition d'images, il s'agit de l'optique, du capteur et de son électronique de lecture. Ces deux étages introduisent des dégradations qu'un étage de traitement supplémentaire doit corriger. Ces trois parties composent le module caméra, qui est de plus en plus distribué comme un composant discret auprès des intégrateurs.

1.1.1.1 Le système optique

Le principe du système optique consiste à focaliser les rayons à mesurer sur une surface sensible. Le système optique est la partie qui fait l'objet du plus d'économies de la part des fabricants, (par exemple en la remplaçant par un sténopé ou une lentille en plastique) car c'est la partie la plus coûteuse du système de capture des images (le prix du module peut décupler en fonction de l'optique qui lui est adjointe) [Global Sources 2009]. La plus grosse part du coût des optiques vient des éléments mécaniques, comme les lentilles mobiles qui permettent de réaliser la mise au point ou de changer de focale (zoomer). La seconde source de coûts est l'intégration d'éléments de différentes technologies que sont les lentilles, le capteur et le boitier. Alors que la taille du capteur est réduite, le positionnement des modules optiques doit faire l'objet de processus particuliers qui augmentent les coûts de production. C'est pourquoi, dans la grande majorité des cas, les intégrateurs cherchent à supprimer ces éléments en utilisant des lentilles réduites à un sténopé amélioré par une lentille, ce qui offre l'avantage de proposer une profondeur de champ élevée. Les systèmes sont conçus pour fonctionner à l'hyperfocale, et donc supprimer toute opération de mise au point. L'utilisation de « wafer level cameras » permet d'intégrer l'optique directement dans le procédé de fabrication du capteur.

1.1.1.2 Les imageurs

Le prix d'un capteur est directement corrélé à sa surface ainsi qu'à sa technologie de fabrication. Depuis quelques années la technologie CMOS [Gamal 2005] a pris le pas sur la technologie Charge-Coupled Device (CCD), essentiellement en raison du coût de fabrication des capteurs dans cette technologie, mais aussi de leur consommation électrique qui peut être 10 fois inférieure à celle des capteurs CCD. De plus, la technologie CMOS rend possible l'utilisation de photodiodes de quelques micromètres de côté, résuisant drastiquement la surface des capteurs et donc leurs coûts de fabrication, mais aussi d'augmenter la résolution pour répondre à la demande du marché. Contrairement à la technologie CCD, où la quasi-totalité de la surface des photosites

est dédiée à la photodiode, la technologie CMOS nécessite l'utilisation de composants complémentaires qui permettent la lecture des valeurs des pixels. Le facteur de remplissage correspond à la surface du photosite effectivement dédiée à la surface photosensible (photodiode) par rapport au reste de l'électronique utilisée pour l'amplification, l'adressage, la lecture, le reset etc. Pour augmenter ce facteur, les fabricants placent des micro-lentilles directement devant le photosite. Ces lentilles focalisent les rayons lumineux sur la surface sensible, ce qui permet d'exploiter la majeure partie de la lumière directement sur le capteur. De plus, comme la sensibilité des capteurs CMOS reste faible pour les raisons évoquées précédemment, le signal de chaque pixel est amplifié. Cette amplification a l'avantage de rendre la lecture de la donnée non destructrice puisqu'elle agit comme un tampon. Les travaux de *Chen* [Chen 2000], puis plus récemment de *Tisse* [Tisse 2008] ont confirmé que la qualité d'image n'était pas améliorée avec l'augmentation de la densité de pixels sur la même surface de capteur. La grande majorité des capteurs actuellement fabriqués et utilisés au sein des systèmes embarqués sont des capteurs de type CMOS en raison de leur faible consommation électrique, et de leur faible coût de fabrication. Cette technologie permet maintenant d'atteindre des pixels de quelques microns (actuellement moins de 1,75 μm) de côté.

L'acquisition des couleurs

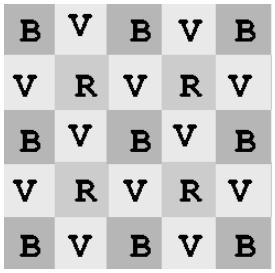


FIG. 1.2: DISPOSITION DU FILTRE DE BAYER.

Comme les photodiodes du capteur ne sont sensibles qu'à l'intensité lumineuse, un filtre de couleur est placé devant chaque diode, en alternant trois couleurs primaires. Le filtre le plus connu est le filtre de *Bayer* représenté en Figure 1.2, mais d'autres arrangements et l'utilisation d'autres couleurs sont proposés par les chercheurs [Gorokhovskiy 2006, Alleysson 2004] comme l'utilisation de pixels panchromatiques comme le propose Kodak [Hamilton 2007]. En prenant l'exemple du filtre de *Bayer* apposé à un capteur qui serait composé de 1000×1000 photosites, il possède en réalité une résolution de 500×1000 pour la couleur verte, une résolution de 500×500 pour les photosites recouverts d'un filtre de couleur rouge, et la même chose pour ceux recouverts d'une filtre de couleur bleue. Nous appellerons les images issues de ces capteurs « images brutes ».

D'autres solutions permettent d'acquérir directement des images couleur à pleine résolution. Il s'agit par exemple d'envoyer la scène sur trois capteurs ayant chacun un filtre de couleur ou d'exploiter *Foveon*, pour lequel chaque photosite intègre les trois couleurs en exploitant les propriétés d'absorption de la lumière par le silicium [El Gamal 2002]. Il suffit d'utiliser plus de pixels bleus que de pixels verts dans le filtre de couleur. Différents arrangements sont régulièrement proposés selon les constructeurs (Fuji, Kodak etc.).

On considère qu'un pixel est blanc lorsque toutes ses composantes sont à 100 % d'intensité, et gris lorsqu'elles sont toutes à une même intensité. Or l'intensité de chaque longueur d'onde, et donc de couleur, varie d'un illuminant à l'autre, ce qui implique que le blanc ne correspond jamais à 100 % d'intensité quel que soit le jeu de couleurs primaires sélectionné. Les coefficients sont connus lorsque la température de l'illuminant est connue et sont généralement normalisés pour les éclairages artificiels, et sont estimés pour des éclairages naturels. Les longueurs d'onde qui composent l'illuminant peuvent être estimées plus ou moins précisément par une simple mesure. Comme le silicium n'a pas la même sensibilité en fonction des longueurs d'onde, l'arrangement du filtre de couleur est essentiel. Par exemple le signal correspondant au bleu nécessite d'être plus amplifié que les autres couleurs, les bruits deviennent donc plus visibles.

1.1.2 Les défauts introduits lors de l'acquisition des images

L'acquisition des images est réalisée au moyen d'un système optique derrière lequel est placé un capteur photosensible qui est lu par une électronique adaptée. Le signal est dégradé au fur et à mesure qu'il passe par chacun de ces dispositifs. Une partie des rayons lumineux diffractés et réfractés est indésirable et produit des défauts optiques qui sont ensuite visibles sur l'image. Ensuite la capture de l'image introduit des bruits, liés à l'électronique mais aussi aux techniques employées pour lire les signaux.

1.1.2.1 Défauts introduits par l'optique

Cette partie ne se veut pas être une liste exhaustive des défauts causés par les différents systèmes optiques, que la littérature couvre déjà largement depuis les débuts de l'optique [Balland 2007, Félice 2009]. Nous considérerons donc essentiellement les problèmes induisant la plupart des défauts dans les images issues de capteurs bas coût.

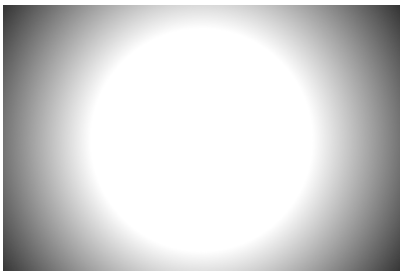


FIG. 1.3: EXEMPLE DE VIGNETTAGE.

Le système optique ne laisse pas passer toutes les fréquences du spectre, généralement limité au visible (pour les applications dans ce domaine) grâce à un filtre passe-bande placé devant le capteur – sans quoi le capteur réagirait avec toutes les longueurs d'onde auxquelles il est sensible (par exemple avec le proche infrarouge). Selon les matériaux utilisés pour la fabrication du système optique les différentes longueurs d'onde ne sont pas absorbées de la même manière en fonction de l'épaisseur des lentilles, ou des défauts liés au procédé de fabrication. Par exemple, l'utilisation de certaines lentilles induit une dominante bleue sur l'image ou encore un vignettage sur les bords de l'image comme l'illustre la Figure 1.3. De même, les propriétés de réfraction et de diffraction varient spatialement pour une optique donnée et sont à l'origine de nombreux défauts, comme les aberrations chromatiques, ou les « franges pourpres » causées par la déviation des rayons des différentes couleurs. Ils causent aussi les déformations géométriques bien connues en coussinet ou en barillet qui peuvent être caractérisées. L'organisation des éléments optique et leur distance par rapport au capteur ont aussi un impact majeur sur ces défauts, comme par exemple, les jeux de diffraction et de réfraction entre les microlentilles généralement placées devant les photosites d'un capteur.

D'un point de vue spatial, le système optique agit aussi comme un filtre passe-bas dont les propriétés varient en fonction de celles des matériaux utilisés (épaisseur, densité, impuretés et position des lentilles etc.) en modifiant leur diffraction. L'image d'un point donne systématiquement une tache floue. Si deux points sont trop proches, les deux taches se chevauchent pour n'en former qu'une seule. La résolution, ou le pouvoir séparateur maximal d'une optique est exprimée en radian. Cette résolution maximale – pour des matériaux parfaits – est inversement proportionnelle au diamètre de l'optique et proportionnelle à la longueur d'onde utilisée et donc à la couleur. Ainsi, l'image acquise par un capteur de résolution supposée infinie est systématiquement floue. Il est donc souhaitable, pour les fabricants, d'utiliser un capteur dont la résolution est inférieure à celle du système optique. Si ce n'est pas le cas, la fonction de transfert de modulation d'une optique permet de caractériser avec précision ces défauts.

Généralement une caractérisation des défauts est réalisée par les utilisateurs eux-mêmes pour chaque dispositif lorsqu'ils sont utilisés à des fins scientifiques (astronomie, télédétection). Pour les applications courantes (photographie et vidéo amateur et professionnelle, sécurité etc.), une caractérisation est réalisée au niveau des

séries de produit, ce qui permet de corriger une grande partie des défauts. Dans le cas des téléphones portables, soit la caractérisation des défauts les plus visibles est réalisée au niveau du produit, ainsi seuls le vignettage, un rehaussement des contours et une correction globale des couleurs sont généralement réalisés, mais parfois, uniquement une correction élémentaire des distortions, soit aucune caractérisation n'est réalisée entraînant des corrections approximatives. Des traitements inverses peuvent être utilisés lorsqu'il s'agit de reconstruire une image à partir des caractéristiques optiques, notamment à partir de la Fonction de Transfert de Modulation (FTM).

1.1.2.2 Les sources de bruits

Quel que soit le procédé d'acquisition, de nombreuses sources de bruits [Hewlett-Packard 1998] viennent détériorer le signal, d'abord au niveau des photosites eux-mêmes, ensuite au niveau de la lecture des signaux issus du capteur [Kodak Eastman Co. 2005]. Nous avons choisi de les regrouper en quatre sources.

- Le bruit spatial fixe

Le bruit spatial fixe est essentiellement causé par la non uniformité des propriétés du substrat, qui rend chaque photodiode unique. Lorsque plusieurs convertisseurs analogique-numérique sont utilisés pour le signal, comme c'est généralement le cas des capteurs CMOS qui en utilisent souvent un par ligne ou par colonne, un bruit caractéristique est visible. On peut voir un exemple de bruit de colonne sur la Figure 1.4, un bruit propre à chaque colonne est ajouté. La valeur de ce bruit additif peut être caractérisée au niveau de chaque pixel. De plus, chaque pixel présente un bruit additif qui lui est propre et qui est causé par les disparités des propriétés du substrat au niveau de chaque pixel, ou des composants qui permettent de les lire. Comme ce bruit est fixe, il peut être aisément caractérisé.

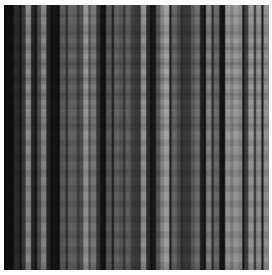


FIG. 1.4: EXEMPLE DE BRUIT DE COLONNE.

- Le bruit de pixel

Le bruit de pixel dont un exemple est visible en Figure 1.5 est composé de différentes sources de bruit. D'abord le bruit thermique *Johnson Nyquist* présent dans tous les composants du capteur, il est causé par l'agitation des électrons qui dépend de la température T, et des propriétés du capteur comme la taille des capacités C, mais aussi du temps d'intégration (ou temps de pose). Il devient prépondérant en condition de faible luminosité et est de la forme kT/C. Le courant de fuite propre à chaque photosite induit un bruit qui est d'autant plus visible que le temps avant la lecture des pixels est élevé. De manière inverse, lorsque la fréquence de lecture des pixels est trop élevée, un bruit de lecture est visible, puisque les capacités ne sont pas totalement vides avant leur chargement par les électrons de l'exposition suivante. Le bruit de photon a une distribution qui répond à une loi de *Poisson*. La racine carrée de son intensité est proportionnelle à la racine carrée de l'intensité des photons reçus par la surface photosensible. De plus, les disparités du substrat induisent le « dark current noise », dont la valeur est aussi proportionnelle à la racine carrée de l'intensité des pixels de l'image. Ces disparités modifient le courant de base de la photodiode qui correspond aux valeurs des pixels noirs, d'où son nom. Entre chaque exposition, ou capture d'une trame, les capacités utilisées pour les charges accumulées dans les photodiodes doivent être vidées. Les capacités ne sont jamais parfaitement déchargées, aussi ce bruit dépend de la capacitance du photosite et de son temps de déchargement. Plus la fréquence de lecture est élevée et plus le temps accordé à vider ces capacités est faible ce qui induit des valeurs parasites qui seront présentes à l'exposition suivante.

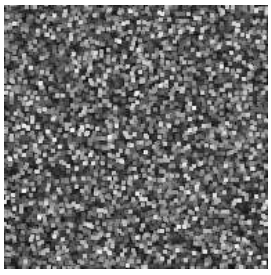


FIG. 1.5: EXEMPLE DE BRUIT DE PIXEL

• Le bruit impulsionnel Le bruit impulsionnel est causé par des imperfections des composants, des erreurs de transmission entre le capteur et les composants de traitement qui peuvent engendrer des valeurs de pixels aberrantes (pixels saturés, noirs, ou simplement modifiés). Il peut être causé par l'effet de certaines imperfections lors du dopage des composants, et se révèle lorsque des durées d'expositions longues sont utilisées. La comparaison des pixels avec les valeurs voisines permet d'en déterminer la présence.

• Le bruit de quantification Le bruit de quantification est visible lorsque l'image n'est pas codée sur une dynamique suffisante. Or, les capteurs actuels utilisent des convertisseurs analogique-numérique généralement supérieurs à 8 bits (10 à 12 bits), ce qui le rend imperceptible lors de l'utilisation directe des images. Toutefois, les expositions réalisées ne permettent généralement pas de couvrir la totalité de la dynamique du capteur.

Conclusion sur l'acquisition des image

Les modules imageurs utilisés dans les dispositifs mobiles intègrent une partie optique, un capteur et l'électronique de traitement en un seul module. Son coût de fabrication dépend directement des dimensions du capteur et du système optique utilisé. Pour réduire ces coûts de fabrication, les optiques sont simplifiées au maximum et la surface silicium des capteurs réduite autant que possible, ce qui a des conséquences sur la qualité des images.

Cette section a présenté les principaux défauts introduits à l'acquisition de l'image, qu'ils soient optiques ou électroniques. Il est donc nécessaire d'ajouter une électronique de traitement afin de reconstruire et d'améliorer les images avant leur utilisation.

1.2 Présentation des algorithmes de reconstruction d'image

Les signaux issus des capteurs formant des images brutes (ou images Color Filter Array (CFA)) ne sont pas directement exploitables en tant qu'image couleur. Cette image est constituée d'une succession de pixels correspondant chacun à l'échantillonnage de l'image par le capteur et le filtre de couleur placé devant chaque pixel. De plus, elle est altérée par différents bruits électroniques ou défauts optiques. C'est pourquoi un ensemble de traitements sont utilisés pour reconstruire et en améliorer l'aspect.

La chaîne de reconstruction et d'amélioration usuelle est présentée dans la première partie. Les traitements de cette chaîne sont ensuite décrits, ainsi que quelques algorithmes utilisés pour les réaliser au sein des dispositifs mobiles actuels. Enfin, nous déterminerons des chaînes de référence qui se basent sur ces algorithmes, et qui seront utilisées tout au long de ce document.

1.2.1 Présentation de la chaîne de reconstruction d'image

La chaîne de reconstruction d'image ne fait pas l'objet de standardisation, ni dans le choix des traitements à réaliser, ni même dans la manière de les enchaîner, comme cela peut être le cas pour certains standards de compression [Hanzo 2007]. Les traitements réalisés dépendent directement des technologies de capture utilisées et des choix des intégrateurs. De plus, chaque intégrateur utilise ses propres variantes algorithmiques qui peuvent représenter une réelle valeur ajoutée pour eux. En effet, il n'est pas rare de voir tel ou tel constructeur mettre en avant une technologie logicielle de rendu des couleurs.

Les traitements de la chaîne de reconstruction d'image peuvent être regroupés en quatre catégories, auxquelles il faut ajouter l'utilisation de l'image elle-même, comme le présente la Figure 1.6. D'abord, les bruits de l'image sont corrigés le plus tôt possible à la sortie du capteur. La présence de bruit dans une image peut fortement dégrader les résultats des traitements exécutés par la suite.

Ensuite, l'image issue du capteur fait l'objet d'un traitement qui adapte sa dynamique en fonction de l'utilisation future. C'est aussi à ce moment que sont corrigés les contrastes et la répartition des tons.

La reconstruction des couleurs, qui vient généralement ensuite, est l'étape la plus importante. D'une part, elle permet d'obtenir une image à pleine résolution, généralement sur trois plans à partir d'une image brute, d'autre part, elle adapte les couleurs de l'image en fonction de celles des illuminants de la scène.

Enfin une étape d'amélioration de l'image est réalisée. Elle consiste essentiellement à rehausser les contours, mais aussi les couleurs ou les contrastes en fonction de l'utilisation visée. L'image ainsi obtenue peut alors être enregistrée, compressée, transmise sur un réseau ou exploitée par une application.

Les traitements employés pour la reconstruction de l'image diffèrent en fonction des technologies de capture mises en œuvre, mais aussi en fonction des applications auxquelles elle est destinée. L'exemple de la Figure 1.7 illustre, en niveaux de gris pour les besoins de reproduction de ce manuscrit, cette reconstruction sur une image photographique. Actuellement, dans la grande majorité des cas, l'image est acquise pour la prévisualisation sur l'écran du dispositif mobile, elle ne fait donc l'objet d'aucun stockage ou réutilisation. Dans ce cas, un traitement partiel qui consiste à reconstruire les couleurs et rehausser les contrastes, sur une image sous-échantillonnée, est généralement suffisant. Lorsque l'image est destinée à une utilisation vidéo, on distinguera le cas Haute Définition (HD) du cas où l'image est compressée et transmise pour une vidéo-conférence. De plus, certaines applications peuvent être notablement accélérées si certains retraitements sont réalisés lors de la reconstruction de l'image, comme par exemple la détection de coins pour une recherche de zones d'intérêt comme les visages des interlocuteurs.

1.2.2 Réduction des bruits et des défauts

La littérature propose de nombreuses méthodes de réduction des bruits [Seo 2007, Motwani 2004], c'est pourquoi nous ne présentons ici que celles susceptibles d'être utilisées dans les systèmes mobiles qui nous intéressent.

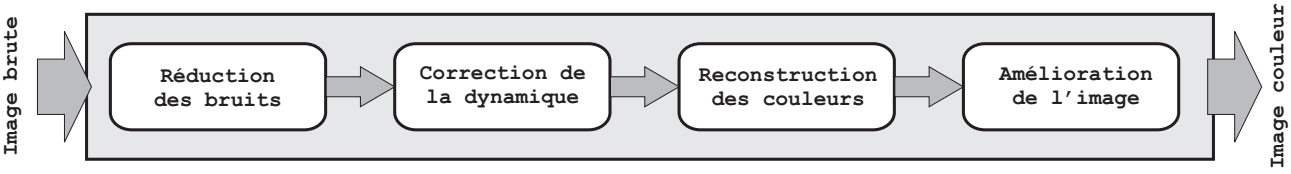


FIG. 1.6: SCHÉMA SIMPLIFIÉ DES CLASSES DE TRAITEMENT DE LA CHAÎNE DE RECONSTRUCTION D'IMAGE.

(a) IMAGE BRUTE ORIGINALE (BRUITS + FILTRE BAYER) (b) IMAGE APRÈS RECONSTRUCTION

FIG. 1.7: IMAGES PHOTOGRAPHIQUES DU *Mont Blanc* RECONSTRUITE (a) IMAGE BRUTE (b) RECONSTRUITE.

1.2.2.1 Correction des défauts fixes et du bruit spatial fixe

Une première étape consiste à corriger les défauts liés à la capture comme le bruit spatial fixe et les pixels « morts ». La suppression du bruit spatial fixe est réalisée par soustraction de l'image de ce bruit préalablement caractérisé. De plus en plus de capteurs intègrent cette fonctionnalité. Certains défauts optiques comme le vignettage [Yu 2004] sont aussi corrigés grâce à la caractéristique du vignettage, c'est à dire son profil radial.

1.2.2.2 Correction du bruit impulsionnel

Le bruit impulsionnel, qui est causé par la présence de pixels aberrants, peut être supprimé en remplaçant leur valeur par une alternative calculée à partir des voisins. Les filtres médians, médians pondérés et apparentés [Bovik 2002, Singh 2003, Hamza 1999, Gil 1993, Brownrigg 1984] sont adaptés pour la suppression de ce type de bruit tout en préservant les contours. D'autres méthodes de traitement non linéaires ou statistiques consistent à vérifier si la valeur du pixel suspect ne s'éloigne pas de celles de ses voisins [Nallaperumal 2006, Smolka 2004], tandis que les méthodes spatio-temporelles consistent à utiliser plusieurs images successives d'une vidéo [Lane 1999, Bosco 2002] pour déterminer si les pixels ont des valeurs aberrantes. L'utilisation d'un tel algorithme au sein de dispositifs embarqués nécessite une estimation des mouvements et l'appareillement des trames successives, ce qui est particulièrement coûteux en calcul. Les méthodes statistiques et apparentées aux filtres médians sont donc largement utilisées aujourd'hui.

1.2.2.3 Correction du bruit de pixel

Comme nous l'avons vu, le bruit de pixel est un bruit blanc additif, dont une des composantes répond à une distribution gaussienne, et l'autre à une loi de *Poisson*. Les approches spatiales sont les plus répandues, il s'agit de moyenner les valeurs des pixels ou d'appliquer une convolution comme par exemple un flou gaussien mais ces méthodes altèrent les contours de l'image. Les méthodes locales adaptatives comme le filtre bilatéral, proposé par *Tomasi* [Tomasi 1998] permet de les préserver. Il peut être vu comme une convolution pour laquelle les valeurs des coefficients sont ajustés à chaque pixel en fonction des valeurs des voisinages du pixel à traiter. Certaines de ces approches caractérisent le bruit directement dans l'image brute pour le corriger par un filtre spatial adapté [Angelo Bosco 2001].

Les méthodes basées sur la transformée en ondelettes sont fréquemment proposées [Gavrincea 2007]. Le principe consiste à supprimer les coefficients d'ondelettes correspondant au bruit, ce qui revient à un seuillage (qui peut être plus ou moins complexe). La détermination de ce seuil est réalisée à partir des caractéristiques du bruit à supprimer. Toutefois ces méthodes sont rarement utilisées au sein de dispositifs embarqués de grande consommation. Ceci est essentiellement lié au fait que les grands fabricants de Digital Signal Processor (DSP) ne proposent pas de librairies de transformées en ondelettes optimisées pour leurs composants [Qureshi 2005] de traitement du signal. Cette méthode nécessite en outre un étage supplémentaire de transformation discrète et son inverse. C'est aussi le cas des méthodes de débruitage dans le domaine de *Fourier* qui nécessitent les algorithmes de transformation, ainsi que la mémoire pour stocker les plans intermédiaires.

1.2.2.4 Correction des défauts optiques

Certains défauts optiques comme le vignettage sont aussi corrigés dès la sortie du capteur. Pour cela, il suffit de disposer d'une image ou du profil radial du vignettage et de le soustraire aux valeurs des pixels en fonction de leur position par rapport au centre de l'image. Cette méthode est valable pour d'autres défauts comme l'aberration chromatique latérale (franges pourpres) lorsqu'elle est caractérisée [Mallon 2007, Kaufmann 2005, Yamashita 2006], cette correction reste toutefois réservée aux dispositifs haut de gamme. Il s'agit avant tout de réaligner les différents plans couleur, ce qui peut être fait directement sur l'image brute. Lorsque'une source lumineuse dans le champ du capteur crée, par des phénomènes de diffraction et de réfraction entres les lentilles, un effet d'éblouissement, alors la quasi-totalité des pixels de l'image reçoit de la lumière parasite. Globalement, son intensité est la même pour tous les pixels de l'image, il suffit donc de la caractériser et de la soustraire à tous les pixels, ou par correction d'histogramme. Les distorsions en barillet et en coussinet peuvent aussi être corrigées dans l'image brute à la sortie du capteur ou sur les différents plans de l'image finale [Brauer Burchardt 2004, Frank 2007] mais elles nécessitent d'être caractérisées. C'est pourquoi des méthodes se proposent de les réaliser sans étape de calibration [Yu 2003a] ce qui permet de réduire fortement les coûts d'intégration. Elles sont toutefois souvent négligées pour les applications de grande consommation, ou réalisées avec des algorithmes simplifiés.

Conclusion sur la réduction des bruits

La correction des bruits requiert quasiment un algorithme par type de bruit. Les défauts fixes doivent être caractérisés pour être corrigés efficacement. Le bruit impulsionnel est généralement corrigé à l'aide de filtres non linéaires spatiaux. Les filtres médians et ses variantes restent les plus largement utilisés. Le bruit de pixel, qui est un bruit blanc additif caractérisé peut être réduit à l'aide de nombreuses techniques. Les solutions spatiales locales adaptatives comme le filtre bilatéral sont privilégiées pour leur simplicité. Enfin les corrections des défauts liés à l'optique ne sont que rarement réalisées sur les dispositifs embarqués à faible coût. Il est probable qu'à l'avenir les réductions du bruit basées sur la transformée en ondelettes soient de plus en plus utilisées.

1.2.3 Correction de la dynamique et des contrastes

Cette étape regroupe plusieurs types de traitements. Les premiers consistent à s'assurer que l'exposition est correctement réalisée et que l'ensemble de la dynamique du capteur est exploitée. Les seconds consistent en l'amélioration des contrastes. Au sens large, la technique permettant d'adapter une image à une dynamique donnée est le « tone mapping » [Matkovic 1997].

Les capteurs possèdent des Convertisseur Analogique-Numérique (CAN) qui sont utilisés pour échantillonner les valeurs des pixels selon autant de nuances convertisseur le permet. De plus, un capteur peut avoir une

courbe de réponse non linéaire. Pour que les données puissent être manipulées elle doivent être converties (standard ISO 14,525) dans un espace plus grand (par exemple 10 bits vers 16 bits). Si un capteur possède un CAN 10 bits, comme c'est généralement le cas, 1024 nuances par pixels pourront être discriminées, ce qui correspond à la dynamique de l'image. Cette dynamique peut être pleinement exploitée seulement si l'exposition est correcte. Prenons par exemple une image fortement sous-exposée, la plupart de ses pixels ont des valeurs faibles, comprises par exemple entre 0 et 50, alors que 1024 niveaux sont disponibles. A l'inverse les valeurs des pixels d'une image sur-exposée seront élevées, et peuvent être par exemple entre 900 et 1024. Là encore, seules quelques nuances sont exploitées.

De plus, la mauvaise utilisation de la dynamique peut provenir d'une scène peu contrastée (un paysage dans le brouillard, une vue de l'océan), un problème d'éclairage, ou encore une mauvaise prise de vue, derrière une vitre ou encore liée à l'utilisation d'une mauvaise optique. Dans ce cas, toutes les valeurs des pixels se concentrent autour d'une valeur et seules quelques nuances sont utilisées. La dynamique des capteurs tend à augmenter et à dépasser celle des moniteurs. C'est pourquoi, il peut être nécessaire de compresser la dynamique au lieu de l'étendre afin de rendre perceptibles des informations qui ne le sont pas. Quelques exemples d'images obtenues à cette étape sont visibles en Figure 1.2.3.2.

1.2.3.1 Contrôle d'exposition

Il est possible d'appliquer une correction directement au moment de la capture des images en agissant sur différents leviers. Ces leviers sont la sensibilité du capteur, c'est à dire le gain des amplificateurs, le temps d'exposition, et l'ouverture du diaphragme de l'optique, le plus souvent fixe pour la plupart des systèmes bas coûts. Le point noir du capteur peut aussi être ajusté, comme son nom l'indique, il permet de faire varier le seuil du noir. Pour quantifier les valeurs de ces paramètres, une première image est généralement lue et mesurée. Si les valeurs de ses pixels sont trop éloignées des extrema des valeurs de que peut prendre la dynamique, il y a mauvaise exposition, alors une commande est envoyée au module optique pour qu'il adapte un des paramètres sus-cités.

Cette correction ne s'applique qu'aux expositions suivantes. Il est donc nécessaire de réaliser l'estimation de ces paramètres en permanence. Pour cela, l'image à pleine résolution n'est pas nécessaire. Elle peut par ailleurs être directement réalisée sur les données de l'image brute sous-échantillonnée. Généralement la correction apportée est globale à tout le système de capture (variation du gain, du diaphragme, du temps de pose), bien que la littérature présente quelques capteurs permettant des corrections par pixel [McCaffrey 2000] ou groupe de pixels. Ces capteurs restent pour l'heure inutilisés dans les produits de grande consommation.

Différentes méthodes ou algorithmes peuvent être utilisés pour le contrôle d'exposition. Ceux-ci étaient déjà largement utilisés à la fin des années 80 sur les appareils reflex argentiques ainsi que les camescopes. Dans les années 90, les reflex argentiques se sont vus dotés de capteurs à faible résolution dédiés à cette tâche, remplaçant alors les cellules simples. La mesure peut être globale à l'image, ou effectuée sur des zones. Par exemple le centre de l'image, ou encore sur un visage ou des régions d'intérêt, si le dispositif est en mesure de détecter ceux-ci. D'une manière générale, la mesure consiste à vérifier si les valeurs des pixels de l'image sont réparties sur l'ensemble de la dynamique [Takahashi 2006].

1.2.3.2 Adaptation de la dynamique et correction des contrastes

Une fois l'image capturée, même lorsque des systèmes de mesure d'exposition perfectionnés sont mis en œuvre, la dynamique n'est souvent pas pleinement utilisée. Il s'agit alors d'étendre la dynamique des images,

cette étape est réalisée en début de chaîne de traitement, afin que le maximum de dynamique soit disponible pour les opérations. Étendre la dynamique a pour conséquence l'augmentation des contrastes de l'image. Au contraire, la compression de la dynamique est généralement réalisée en fin de chaîne. Ainsi tous les traitements peuvent exploiter les informations contenues dans l'image. La compression de dynamique est utilisée pour produire des images dont la dynamique est inférieure à celle utilisée pour la capture ou le traitement. Par exemple, une image peut être codée sur 12 bits, et ramenée sur 8 bits au moment de sa compression au format Joint Photographic Experts Group (JPEG). Dans certains cas, il peut s'avérer utile de compresser cette dynamique en début de chaîne pour réduire la taille des données à manipuler.

Les approches globales consistent à déterminer une même correction à appliquer à l'ensemble des pixels de l'image. Elles sont souvent basées sur l'analyse d'histogrammes et des approches statistiques [Bovik 2002]. Des approches locales adaptatives basées sur les histogrammes peuvent être utilisées [Yu 2003b, Stark 2000, Leszczynski 1989, LEE 1980]. Des modèles bio-inspirés [Meylan 2007] comme *Retinex* [Meylan 2006] font l'objet de beaucoup de travaux et dépassent le cadre de cette étude. On peut en voir un exemple en Figure 1.8d. Ce sont aussi des approches locales qui consistent à calculer la valeur du pixel à partir de son voisinage qui peut être de plus ou moins grande taille. Pour *Retinex*, des fenêtres de plusieurs dizaines ou centaines de pixels de côté peuvent s'avérer nécessaires pour obtenir de bons résultats [Ciurea 2004, Zia-ur Rahman 2004]. De nombreuses propositions moins coûteuses en ressources de calcul existent [Sakaue 1995, Castrorina 2006, Artusi 2002]. De telles approches sont de plus en plus souvent intégrées aux dispositifs mobiles puisqu'elles permettent une amélioration notable de l'image sans intervention sur le matériel.

D'autres techniques spatio-temporelles utilisent plusieurs images successives pour affiner [Artusi 2001, Bennett 2005] le rendu final, mais elles nécessitent des capteurs rapides et une mémoire d'image. D'autres approches basées sur la logique floue [Cheng 2000] ou des approches multi-échelles sont proposées [Jin 2001, Dippel 2002] mais restent cantonnées à des domaines d'applications spécifiques. Il reste évidemment possible d'utiliser conjointement les différentes approches. Ainsi, des techniques se proposent de réduire les bruits ainsi que d'adapter la dynamique des images proposées comme [Malm 2007, Petschnigg 2004].

(a) IMAGE ORIGINALE (b) IMAGE SOUS EXPOSÉE (c) IMAGE SOUS-CONTRASTÉE (d) IMAGE APRÈS TRAITEMENT *Retinex*

FIG. 1.8: EXEMPLE DE VUES DU PHARE (a) ORIGINALE, (b) (c) PRÉSENTANT UNE MAUVAISE RÉPARTITION DES TONS ET (d) TRAITÉE PAR L'ALGORITHME *Retinex*.

1.2.3.3 Conclusion sur la correction de dynamique

Nous avons vu au travers de quelques exemples que la littérature foisonne d'approches différentes pour réaliser l'adaptation des images à une dynamique donnée. D'abord le contrôle d'exposition joue un rôle essentiel et est réalisé par analyse statistique de l'image, ensuite, différentes approches permettent de corriger les images. Les plus simples sont les corrections gamma, puis d'histogramme [Bovik 2002, Russ 2002], largement utilisées aujourd'hui. Par ailleurs, les méthodes issues de la théorie *Retinex* ont montré qu'elles sont en mesure de réaliser cette adaptation. Son approche consiste à exploiter les valeurs du voisinage proche et éloigné d'un pixel pour en calculer la valeur. Cette méthode est largement utilisée dans les filtres dits locaux adaptatifs puisqu'ils exploitent les valeurs du voisinage pour réaliser une correction. Il peut s'agir d'une correction d'histogramme locale ou encore une correction gamma dont le paramètre varie en fonction du voisinage. D'une manière générale, toute correction globale peut être exprimée sous forme locale.

1.2.4 La reconstruction des couleurs

La reconstruction des couleurs est l'étape la plus importante de la chaîne de reconstruction des images. Elle permet d'obtenir trois plans couleur à partir des images brutes, mais aussi de corriger les tons de chaque plan couleur afin que l'image présente des couleurs naturelles. Nous distinguons deux étapes distinctes. La première consiste à corriger « la balance des blancs » pour que les couleurs paraissent naturelles, la seconde consiste à « démosaïquer » l'image brute en trois plans couleur de même résolution.

1.2.4.1 La correction de la balance des blancs

En effet, la couleur des éclairages varie en fonction de leur nature, par exemple un éclairage au tungsten sera plus jaune qu'un éclairage au néon quant à lui bleuté. Le soleil à son zénith ne présente pas les mêmes teintes qu'à son coucher où il est nettement plus orangé. Ceci est dû au fait que les longueurs d'onde produites par ces éclairages ne le sont pas toutes avec la même intensité. C'est pourquoi les scènes capturées sous différents illuminants paraissent avoir des « teintes » différentes. Pourtant l'œil humain est sensible à ces différences mais nous ne le percevons pas ou peu, grâce à notre cortex visuel qui est capable de corriger ces informations pour que le « blanc » nous apparaisse blanc quelles que soient les conditions de lumière. C'est l'origine de la théorie *Retinex* [Land 1971] dont nous avons fait référence auparavant. Le capteur n'est pas en mesure de réaliser cette correction, outre la sélection manuelle de l'illuminant, différentes propositions ont été faites pour déterminer automatiquement et à partir de la seule image la valeur des coefficients à appliquer à chaque composante couleur pour restaurer des couleurs naturelles. La recherche est particulièrement active dans ce domaine liant sciences cognitives et traitement du signal.

La plupart des dispositifs utilisent des approches globales, c'est à dire que les coefficients sont déterminés pour l'ensemble d'une image. Elles s'avèrent généralement suffisantes pour la plupart des applications. Toutefois des travaux proposent d'autres méthodes [Agarwal 2006]. Certaines se basent sur une analyse statistique de l'image, d'autres sur la « théorie du monde gris » [Buchsbaum 1980] ou du maximum de réflectance considérant respectivement que l'illuminant peut être approximé par la moyenne des pixels d'une image, ou par la recherche du point le plus réfléchissant (autrement dit le plus clair) [yan Huo 2006]. De nombreuses autres méthodes qui dépassent le cadre de ce document sont proposées, elles sont basées sur des approches statistiques, sur la théorie *Retinex*, sur l'approche GAMUT qui cherche à résoudre un problème contraint, ou encore sur des méthodes d'apprentissages à base de réseaux de neurones [Karungaru 2003]. La plupart des méthodes sont particulièrement calculatoires.

Certaines approches multi-echelles [Li 2007] ou locales [Ching-Chih Weng Chen 2005, Ebner 2009, Gijsenij 2006] permettent d'améliorer les résultats et de tenir éventuellement compte d'éclairages locaux à une scène. Il s'agit d'appliquer la méthode globale à des zones locales de l'image pour en déterminer la correction à apporter. Aucune méthode ne permet d'obtenir une image reflétant fidèlement la réalité, certaines méthodes permettent d'obtenir des images visuellement agréables mais dont les couleurs sont ne sont pas fidèles à la scène. C'est pourquoi certaines combinent différentes techniques [Lam 2005, van de Weijer 2007, Bianco 2007] pour déterminer la couleur de l'illuminant. Là encore, le coût calculatoire peut s'avérer rédhibitoire sur un système embarqué.

Généralement les techniques employées au sein des systèmes embarqués se basent sur un compromis entre la méthode du monde gris et du maximum de réflectance. En effet, ces méthodes permettent de réaliser avec un minimum de calcul et de mémoire d'image une estimation convenable des coefficients de correction à appliquer aux composantes de l'image. De plus, ces méthodes peuvent aisément être dérivées en méthodes locales adaptatives [Provenzi 2008], ou donner naissance à d'autres algorithmes propriétaires plus complexes.

1.2.4.2 Le démosaïquage

Le démosaïquage calcule trois plans couleur à pleine résolution à partir de l'image brute. Chaque pixel du capteur est placé sur un plan correspondant à sa couleur, les plans ainsi constitués sont incomplets, comme on peut le voir sur la Figure 1.9. C'est pourquoi, un processus d'interpolation est nécessaire pour déterminer la valeur des pixels manquants. Au regard du volume d'information qui est à reconstruire ($2/3$ des informations de l'image finale) on comprend l'importance de cette étape. De nombreux travaux [Alleysson 2004, Gunturk 2005] visent à proposer des algorithmes de reconstruction précis, leur présentation dépasse le cadre de cette thèse. Par ailleurs, des alternatives aux filtres de couleurs usuels sont régulièrement proposées par les fabricants de capteurs. Celles-ci ne seront pas abordées ici, nous retiendrons toutefois que l'algorithme utilisé dépend directement de la structure de la nature du filtre utilisé, et que l'étape de démosaïquage devient inutile lorsque le dispositif de capture est en mesure de fournir directement des images à pleine résolution.

La reconstruction de ces pixels manquants induit inévitablement des artefacts qu'il convient de limiter. Ils sont de plusieurs natures et vont d'un simple effet flou, du moiré à l'effet « zipper ». Si des algorithmes d'amélioration permettent de les réduire ultérieurement il est souvent préférable de les limiter dès l'étape de démosaïquage.

Les méthodes linéaires les plus simples consistent à recopier la valeur du plus proche voisin, ou à les moyenner pour calculer les pixels manquants par démosaïquage bilinéaire. Les contours sont altérés par l'utilisation de ces méthodes, c'est pourquoi des améliorations sont proposées [Malvar 2004]. Des efforts sont mis a la préservation de ces contours, la méthode la plus connue est celle de *Hamilton et Adams* [Hamilton 1997, Adams 1997] qui pré-calcule les gradients de l'image. De nombreuses approches exploitent les corrélations entre composantes couleur comme la méthode de la constance de teinte [Cok 1987] ou plus récentes [Gunturk 2002, Gupta 2001, Lukac 2004]. Le filtre bilatéral qui préserve les contours d'une image trouve là aussi une utilisation [Akuiyibo 2006]. Des méthodes bio-inspirées sont aussi proposées [Alleysson 2005]. Ces méthodes spatiales sont largement employées dans les systèmes embarqués. Les intégrateurs proposent souvent leur propres algorithmes dérivés de ces dernières, préférant généralement l'utilisation des méthodes peu coûteuses dont les artefacts sont corrigés ultérieurement lorsque cela s'avère nécessaire.

De nombreuses autres approches voient le jour, mais ne sont pas intégrées au sein des dispositifs embarqués (souvent par manque de mémoire de travail ou de capacité de calcul). Des techniques spatio-temporelles ex-

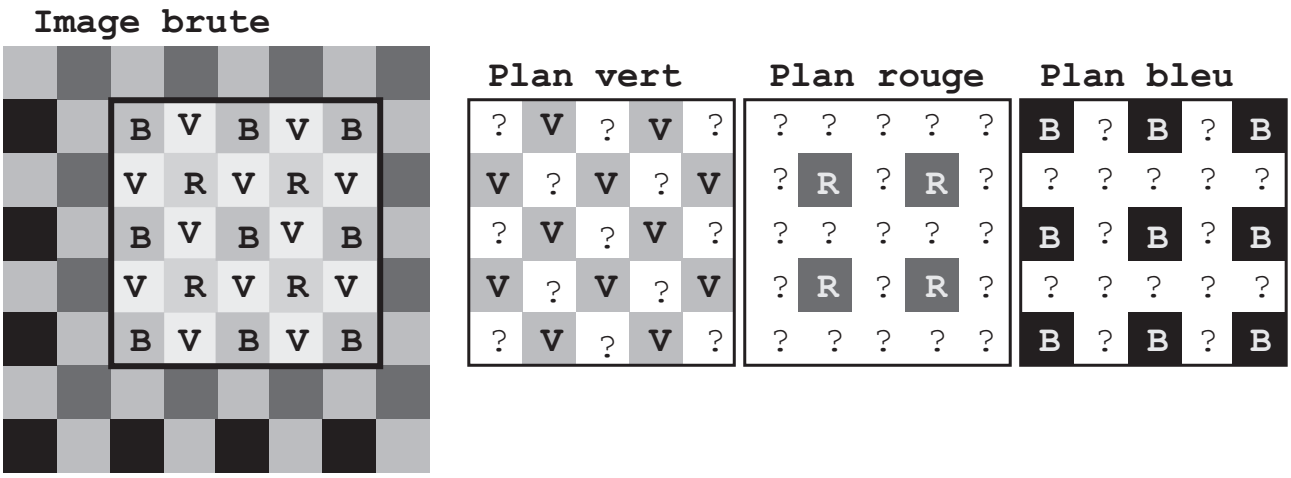


FIG. 1.9: PIXELS MANQUANTS (REPRÉSENTÉS PAR « ? ») LORSQUE CHAQUE PIXEL DE L'IMAGE BRUTE EST ASSOCIÉ AU PLAN COULEUR LUI CORRESPONDANT.

ploitent plusieurs images successives pour déterminer les valeurs des pixels manquants [Farsiu 2006, Wu 2006, Lukac 2005] et des approches par les ondelettes offrent des alternatives aux solutions spatiales [Menon 2007], tout comme les méthodes fréquentielles [Dubois 2005] qui récupèrent les données de luminance et de chrominance par filtrage passe-bande et coupe-bande. Certains proposent de réaliser la réduction des bruits et le démosaïquage de manière conjointe [Hirakawa 2006] ou même le démosaïquage et la répartition des tons [Alleysson 2006], et d'autres algorithmes peuvent être intégrés [Zhang 2007] à ces approches.

Conclusion sur la reconstruction des couleurs

L'étape de reconstruction des couleurs est la plus importante de toute la chaîne de traitement d'image. D'abord elle permet d'obtenir un rendu couleur le plus fidèle possible à la réalité par la balance des blancs, mais elle permet surtout d'interpoler les deux tiers des valeurs des pixels de l'image finale à pleine résolution, puisque seul un tiers de l'information est capturée par le capteur.

Si la correction de la balance des blancs consiste généralement à appliquer des coefficients de correction à chaque composante couleur, leur estimation s'avère un problème complexe. Pour cela, des solutions simples sont généralement employées au sein des dispositifs embarqués offrant un compromis entre fidélité des couleurs et complexité algorithmique.

Le démosaïquage fait l'objet de nombreux travaux, il existe donc une très grande variété d'algorithmes qui peuvent être adaptés au filtre de couleur placé devant le capteur. Les méthodes spatiales sont largement employées dans les systèmes embarqués pour leur simplicité d'intégration. D'une manière générale, c'est l'utilisation des méthodes peu coûteuses dont les artefacts sont corrigés ultérieurement qui est le plus fréquemment rencontrée au sein des dispositifs mobiles. Les intégrateurs proposent souvent leur propres algorithmes dérivés des approches bilinéaires, de la constance de teinte de *Cok* ou de *Hamilton et Adams* simplifiées. Ils peuvent là aussi apporter une réelle valeur ajoutée à la qualité de leurs images par rapport à la concurrence sans intervenir sur les composants de capture.

1.2.5 L'amélioration d'image

La dernière étape consiste à améliorer l'aspect visuel de l'image reconstruite. A ce stade, les bruits sont supposés réduits, les contrastes améliorés et les couleurs reconstruites fidèles à la réalité. Toutefois les traitements qui ont permis d'en arriver à ce stade ont pu introduire des artefacts, nous citerons entre autres un effet de flou de l'image, de moiré comme on peut le voir en Figure 1.10, d'effet de crénelage qui forme des escaliers et de décalages horizontaux et verticaux le long des contours. Ces artefacts peuvent induire des fausses couleurs lorsque les trois composantes sont concernées.

1.2.5.1 Changement d'espace colorimétrique

De nombreux algorithmes de traitement d'image sont généralement exécutés autre que l'usuel rouge vert et bleu qui est souvent produit par le démosaïquage (notons d'ailleurs que différentes représentations pour cet espace existent). D'autres espaces colorimétriques peuvent être utilisés [Jack 2007], et font l'objet de standardisations.

Nous retiendrons seulement que la conversion d'un espace colorimétrique à un autre est réalisée par une multiplication de matrice pixel à pixel [Bovik 2002], généralement en passant par un espace intermédiaire. Les informations de luminance et de chrominance sont souvent séparées pour la plupart des traitements. Les informations spatiales étant stockées dans la chaîne de luminance, de nombreux algorithmes réalisent leur traitement uniquement sur cette chaîne. La correction des couleurs quant à elle est réalisée dans les chaînes de chrominance.

1.2.5.2 Réduction des artefacts

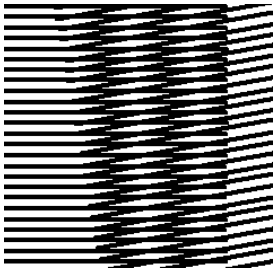


FIG. 1.10: EXEMPLE DE MOIRÉ.

La réduction de fausses couleurs peut être réalisée par un simple filtrage passe-bas dans les chaînes de chrominance, mais les approches sont nombreuses et bien souvent calculatoires [Blinn 1998, Wu 1991, Turkowski 1990] et pour la plupart destinées à la synthèse d'image. Les moins complexes d'entre elles consistent à appliquer un masque de convolution ad-hoc. Il en va de même pour le moiré et les effets de crénelage mais dans l'espace de luminance. Toute la difficulté de cette approche réside dans le calcul des coefficients qui doit être un compromis entre la perdre d'informations de haute fréquence et la réduction des artefacts. Des approches locales adaptatives permettent d'en améliorer les résultats. Les algorithmes qui réalisent ces corrections sont donc souvent propriétaires et adaptés à la chaîne qui est placé en amont.

1.2.5.3 Le rehaussement des contours

Nous avons pu voir que la succession des traitements réalisés a tendance à réduire les contours de l'image. C'est pourquoi une étape de rehaussement des contours est généralement appliquée, elle permet d'augmenter les composantes haute fréquence de l'image. Le rehaussement est le plus souvent appliqué sur les directions horizontales et verticales auxquelles l'œil est le plus sensible. Il est généralement appliqué sur la seule chaîne de luminance, ou sur les trois composantes couleurs.

Les filtres construits autour de noyaux de convolution de *Roberts Prewitt* et *Sobel* sont souvent utilisés ainsi que le calcul laplacien en deux dimension. Mais la méthode la plus employée, lorsque son implémentation est possible, et qui offre le plus de souplesse reste la technique du masque flou qui consiste à soustraire à

l'image originale son pendant flou. Le résultat est seuillé puis additionné à l'image originale réhaussant les contours tout en réduisant certains bruits. De nombreuses variations sont issues de cette méthode [Polesel 2000, Ramponi 1996]. Des approches conjointes avec le démosaïquage [Saito 2005] ou d'autres algorithmes de la chaîne de reconstruction sont aussi proposées [Buyue Zhang 2008].

Si la fonction de transfert de l'ensemble du système situé en amont peut être caractérisée avec précision, alors la déconvolution [Russ 2002] permet d'obtenir l'image originale. Il ne s'agit donc pas d'une technique d'amélioration mais de restauration. Cette approche peut être exprimée très simplement dans le domaine fréquentiel, son expression spatiale est réalisée par des convolutions, dont la taille des noyaux est d'autant plus importante qu'est la dégradation. La déconvolution aveugle consiste à appliquer la même technique sans connaissance de la fonction de transfert. Cette approche, largement utilisée en imagerie scientifique (astronomie, microscopie, télédétection) n'est généralement pas embarquée au sein de dispositifs mobiles qui privilégient les approches spatiales précédemment décrites.

1.2.5.4 Le rehaussement des contrastes et l'amélioration des couleurs

Le rehaussement des contrastes qui peut être réalisé ici repose sur les mêmes techniques que celles vues précédemment en 1.2.3.2. Les méthodes de correction des couleurs vues en 1.1.1.2 sont applicables ici. De plus, les couleurs peuvent être plus ou moins saturées selon les besoins de applications.

Conclusion sur la chaîne de reconstruction et de traitement d'image

Nous avons vu que la chaîne de reconstruction d'image est composée de quatre groupes de traitements.

Les quatre groupes de traitements présentés sont :

1. la réduction des bruits qui consiste à supprimer les artefacts liés à la capture de l'image, comme les bruits électroniques mais aussi un certain nombre d'altérations liées aux optiques utilisées ;
2. l'adaptation de la dynamique et la correction des contrastes, ce qui permet d'obtenir des images équilibrées en corrigeant les erreurs d'exposition, mais aussi en améliorant la répartition des tons ;
3. la reconstruction des couleurs qui est incontournable puisqu'elle permet d'obtenir des images couleur exploitables à partir des données brutes du capteur, cette étape est primordiale pour obtenir des images à pleine résolution à partir de capteurs utilisant un filtre de couleur ;
4. l'amélioration d'image au sens large, permet de réduire les artefacts causés par les traitements précédents mais aussi de rehausser des contours.

Une infinité de chaînes algorithmiques peut donc être présentée tant les approches pour la reconstruction et l'amélioration sont nombreuses.

Les traitements intégrés au sein des dispositifs mobiles de grande consommation sont généralement des traitements simples qui s'adaptent bien aux contraintes de l'embarqué. La réduction des bruits est généralement réalisée à l'aide de filtres médians et gaussiens. L'adaptation de la dynamique est réalisée à partir d'analyses d'histogrammes, mais les filtres locaux adaptatifs sont de plus en plus utilisés pour le rehaussement des contrastes. La reconstruction des couleurs est réalisée par des algorithmes spatiaux, généralement linéaires, parfois une composante non linéaire est ajoutée pour les rendre robustes aux contours. Enfin l'amélioration d'image consiste généralement à rehausser les contours par un filtrage passe-haut après avoir corrigé les défauts induits par les précédents algorithmes de reconstruction. D'autres corrections sont parfois introduites à ce niveau comme une seconde adaptation de la dynamique ou de la répartition des tons, ou une correction des couleurs. En effet, même si les traitements que nous avons proposés sont regroupés en quatre catégories, leur

enchaînement fait l'objet de nombreuses variations qui dépendent des besoins de l'application et des compromis choisis par les fabricants.

Cette diversité permet aux constructeurs d'apporter une importante valeur ajoutée à leurs dispositifs, pourvu que les ressources matérielles permettent l'utilisation des ces algorithmes, tant en termes de ressources de calcul, d'éléments de mémorisation que de données issues du capteur. C'est pourquoi des ressources de calcul flexibles et performantes doivent être proposées afin de supporter cette diversité de traitement.

1.3 Présentation de chaînes d'amélioration d'image

Comme nous avons pu le voir, une grande variété d'algorithmes sont disponibles pour réaliser la reconstruction et l'amélioration d'image. Cette chaîne ne fait l'objet d'aucune standardisation, et sa conception est laissée à la discrétion des intégrateurs qui peuvent la composer en fonction des ressources de calcul et de mémorisation disponibles mais aussi des défauts à corriger et des contraintes à respecter (coûts, temps de traitement etc.).

Pour les besoins de ce manuscrit nous proposons trois chaînes de traitement, chacune correspondant à des choix algorithmiques possibles. Ces chaînes sont avant tout destinées à être utilisées comme « chaînes de référence » pour la suite des travaux présentés dans ce document, bien qu'elles ne constituent aucune référence. Elle ne peuvent donc être directement utilisées sur un dispositif mobile sans avoir préalablement fait l'objet d'ajustements afin de corriger les bruits et déformations qui lui sont propres.

Cette section présente donc brièvement les choix qui ont été réalisés pour leur composition. Les deux premières chaînes algorithmiques présentées traitent les images dans l'espace *RGB* tandis que la troisième dans l'espace de luminance/chrominance.

1.3.1 Première chaîne de traitement

Cette première chaîne de traitement utilise des algorithmes simples, qui limite sa complexité. Elle est schématisée en Figure 1.11 où apparaissent les quatre étapes élémentaires décrites précédement.

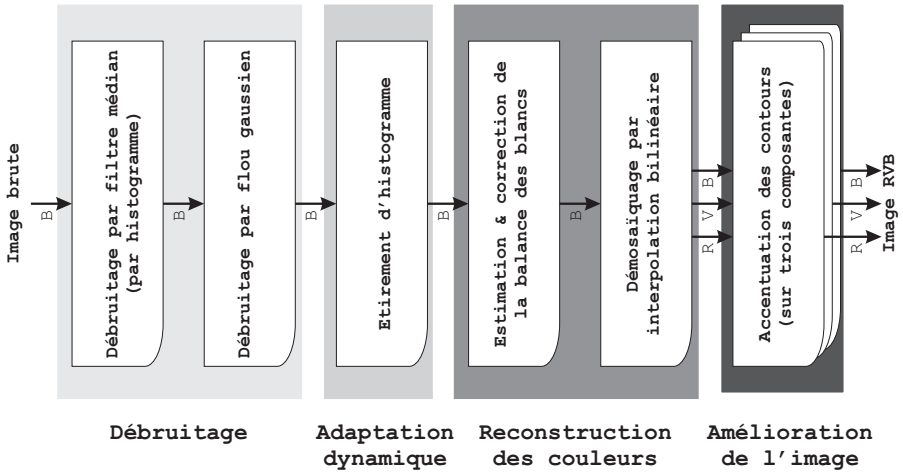


FIG. 1.11: PREMIÈRE CHAÎNE DE RÉFÉRENCE POUR LA RECONSTRUCTION D'IMAGE.

• Correction des bruits Le bruit impulsionnel est corrigé directement dans la matrice de l'image brute grâce à un filtre médian, typiquement de taille 3×3 ou supérieure. Ce dernier est adapté pour ne sélectionner que les pixels correspondant à la composante à traiter. De plus, pour limiter son coût calculatoire il peut être implémenté à l'aide d'histogrammes cumulés. Le bruit de pixel est corrigé sur chaque composante de l'image brute par un flou gaussien 3×3. Comme précédemment, ce filtre est adapté pour traiter l'image brute.

• Adaptation de la dynamique et des contrastes L'adaptation de la dynamique est réalisée par un étirement de l'histogramme de l'image ce qui permet de couvrir l'ensemble de la dynamique disponible.

• Reconstruction des couleurs L'estimation de la balance des blancs est réalisée par la méthode du « monde gris » qui permet d'obtenir les coefficients de correction à appliquer à chaque composante de l'image. Elle peut être directement appliquée à une image brute à condition de tenir compte du format du filtre de couleur. Le démosaïquage bilinéaire est choisi pour sa simplicité, et sont effet passe-bas qui permet de réduire naturellement certains bruits restants. A partir de cette étape l'image obtenue possède trois composantes, rouge verte et bleue.

• Amélioration de l'image Les trois composantes de l'image font l'objet d'une accentuation des contours par addition de l'image avec sa convoluée par un filtre de *Sobel* sur une voisinage 3×3 ou 5×5.

1.3.2 Seconde chaîne de traitement

La seconde chaîne de reconstruction d'image que nous avons constituée est visible en Figure 1.12. Elle est similaire à la première à ceci près que des algorithmes locaux adaptatifs ont été utilisés. D'abord une partie du débruitage est réalisée par un filtre bilatéral robuste aux contours 5×5 après avoir supprimé le bruit impulsionnel par filtrage médian. Ensuite une correction gamma locale adaptative [Sakaue 1995] est utilisée pour améliorer localement la perception des tons. Enfin, l'algorithme de démosaïquage utilisé est semblable à celui proposé par *Hamilton et Adams* [Hamilton 1997] pour une meilleure interpolation au niveau des contours.

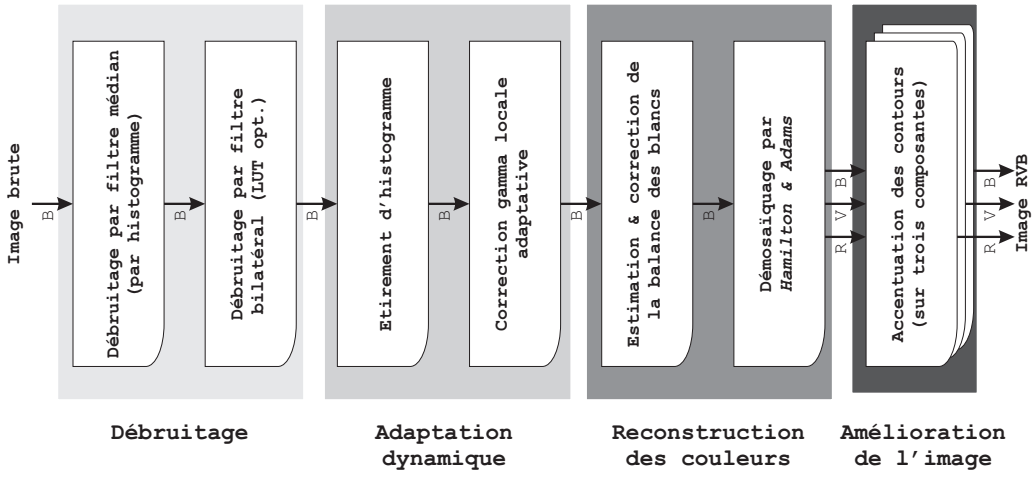


FIG. 1.12: SECONDE CHAÎNE DE RÉFÉRENCE POUR LA RECONSTRUCTION D'IMAGE.

1.3.3 Troisième chaîne de traitement

La troisième chaîne de traitement se propose de réaliser la plupart des traitements sur la chaîne de luminance uniquement. Pour cela, les couleurs de l'image sont reconstruites le plus tôt possible avant d'être converties dans un espace de couleur adapté. On retrouve les algorithmes de la première et seconde chaîne de traitement répartis différemment.

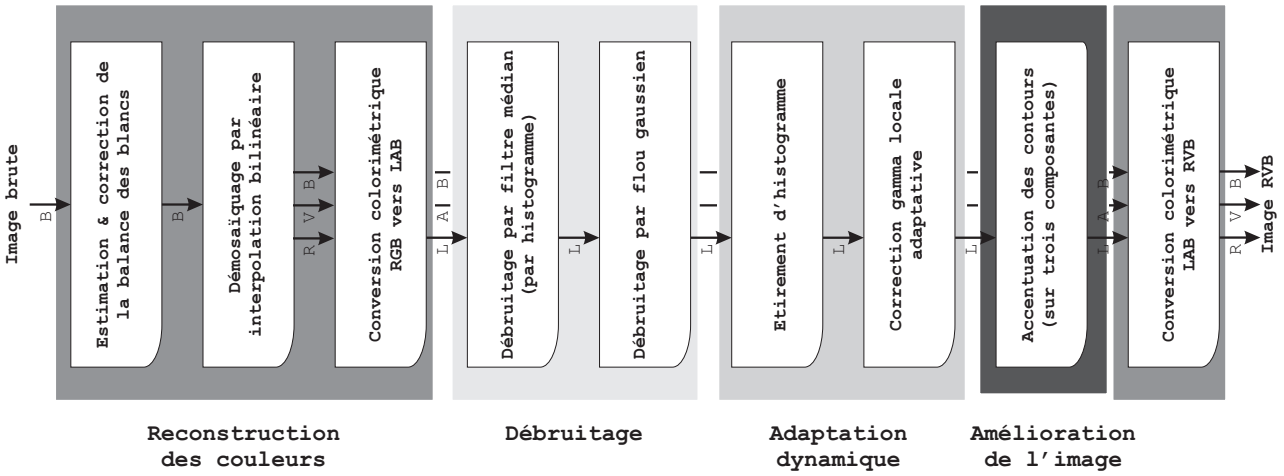


FIG. 1.13: TROISIÈME CHAÎNE DE RÉFÉRENCE POUR LA RECONSTRUCTION D'IMAGE.

Conclusion sur les chaînes de référence

Cette section nous a permis de proposer trois chaînes de traitement qui utilisent certains algorithmes présentés en 1.2.1. Ces trois exemples permettent d'appréhender combien la conception d'une chaîne de traitement peut s'avérer complexe et fait intervenir de nombreux algorithmes et paramètres qui dépendent directement de la nature de l'imageur et des optiques employées.

Les trois chaînes présentées ici seront utilisées par la suite dans l'évaluation de l'architecture. En effet, la nature différente des algorithmes (linéaires, locaux adaptatifs, non-linéaires) ainsi que les approches différentes présentées (RVB, LAB) sont révélatrices du potentiel algorithmique disponible qu'il est nécessaire de supporter.

Conclusion

Les fabricants cherchent à réduire les coûts de fabrication de leur modules d'acquisition d'image. Pour cela, ils intègrent l'optique, le capteur et l'électronique de traitement au sein du même composant. L'optique formant la partie la plus coûteuse du module, elle est généralement simplifiée afin d'être aisément industrialisable. Le prix de revient du capteur est quant à lui directement lié à sa surface. Alors que les consommateurs demandent des produits ayant une résolution la plus élevée possible, la taille des pixels doit diminuer et par là même la sensibilité des capteurs. L'évolution des technologies d'intégration permet d'assurer aujourd'hui de produire des capteurs avec des pixels dont la taille est proche du micromètre. Ces modules doivent donc faire l'objet d'une conception conjointe entre le système optique, le capteur et les traitements de reconstruction d'image.

La correction de l'image se déroule en quatre classes de traitements distinctes. La première consiste à réduire les bruits et les défauts de l'image capturée. La seconde consiste à étendre la dynamique de l'image et en rehausser les contrastes. La troisième est la plus importante, elle a pour but d'obtenir une image couleur à pleine résolution à partir de l'image brute. Enfin une dernière étape d'amélioration d'image offre la possibilité de réaliser des corrections supplémentaires afin d'en améliorer la perception.

Pour chacun de ces traitements, la littérature propose de nombreux algorithmes efficaces. On distingue généralement les approches spatiales, fréquentielles, basées sur la théorie des ondelettes et les approches temporelles. Une grande variété de méthodes existent, et les techniques s'appuyant sur plusieurs approches augmentent le champ des possibilités. Les algorithmes de traitement d'image sont en constante évolution et une infinité de chaînes d'amélioration d'image est possible.

Malgré cela, seules les approches spatiales restent les plus utilisées au sein des dispositifs mobiles de traitement d'image. Même en se basant sur ces approches, nous avons vu, au travers de trois chaînes algorithmiques que nous proposons d'utiliser au long de ce manuscrit, qu'il est possible d'obtenir de nombreuses variations de la chaîne de traitements. Les dispositifs capables de supporter une telle variété de traitement doivent donc avoir un niveau de flexibilité élevé.

CHAPITRE 2

Etat de l'art des processeurs de traitement d'image et vidéo

Où sont présentées différentes architectures adaptées au traitement vidéo.
Où des tendances architecturales sont esquissées à partir de l'étude de ces solutions.

Introduction

ALORS que les applications de traitement vidéo HD nécessitent plusieurs milliards d'opérations par seconde pour être exécutées [Illgner 1999], la surface silicium consentie pour l'amélioration d'image est de quelques millimètres carrés pour quelques centaines de milliwatts de consommation électrique [Talla 2004]. La contrainte de surface silicium est liée au coût de fabrication du circuit qui ne doit pas dépasser quelques dizaines de centimes d'euros. La contrainte de consommation électrique est liée aux capacités des batteries supportant le traitement et la transmission de données sur une longue durée.

Cette contrainte en surface ne permet pas d'intégrer une mémoire d'image au circuit. En effet, en utilisant une technologie d'intégration actuelle comme la TSMC 65 nm [TSMC 2007], il n'est possible de mémoriser que quelques dizaines de lignes d'images couleurs à une résolution [1] HD 1080p sur une surface d'un millimètre carré. Autrement dit, les systèmes basés sur l'utilisation d'une mémoire d'image sont inapplicables au regard de ces contraintes. C'est pourquoi, la littérature propose des architectures matérielles [Dasu 2002] qui permettent de réaliser les opérations de traitement vidéo en temps réel, et notamment d'amélioration et de reconstruction d'images issues de capteurs CMOS ou CCD, d'abord à de faibles résolutions (QCIF, VGA D1 etc.), puis à des résolutions HD 1080p ou pour le traitement d'images photographiques. L'étude de ces architectures nous permet de mettre en évidence certaines solutions architecturales, puis de les retenir ou non en fonction de leur pertinence.

Dans un premier temps, les avantages et les limites des architectures dédiées sont évoqués au travers de quelques exemples. Dans un second temps, d'autres exemples d'architectures reconfigurables sont présentés, puis nous nous intéressons ensuite aux composants programmables en commençant par les plus spécialisés pour terminer par les plus généralistes.

2.1 Présentation des architectures dédiées

Les solutions dédiées permettent d'atteindre une surface silicium et une consommation électrique minimale. Toutefois, les traitements qui sont implémentés restent figés à ceux qui ont été préalablement câblés au sein des composants ou des Propriétés Intelectuelles (PIs).

1. quelques dizaines de lignes correspondent environ à une milliseconde de vidéo HD 1080p.

Dans un premier temps nous présentons quelques composants dédiés et leur performance en termes de surface et de consommation. Afin de réduire le nombre de composants du circuit, ces accélérateurs peuvent être proposés sous la forme de composants discrets, de coprocesseurs ou de PIs que les fabricants peuvent intégrer à leur ASIC. Leur utilisation est présentée dans la seconde partie de cette section. Enfin la troisième partie traite des plate-formes intégrées, où les composants dédiés au traitement d'image trouvent leur place auprès de différents accélérateurs et composants programmables dans un System on Chip (SoC) complet.

2.1.1 Les traitements câblés

Les traitements spécifiques à la chaîne de traitement d'image, bien que la plupart d'entre eux ne soient pas standardisés, ont fait l'objet de nombreuses implémentations dédiées. C'est par exemple le cas du démosaïquage [Garcia-Lamont 2008, Chen 2007, Hsia 2006], du débruitage [Katona 2006, Chen 2008b, Joshi 2006], de la Sum of Absolute Differences (SAD) [Stamatis 2001], mais aussi de la convolution [Eun 1998, Perri 2007], du changement d'espace colorimétrique [Wang 2004], la correction des distortions optiques [Qiang 2006] et de diverses opérations de filtrage [Hernandez 2006]. Le domaine de la compression d'images est particulièrement actif avec des implémentations d'opérations comme la Discrete Cosine Transform (DCT) [Elhamzi 2008, Agostini 2001], mais aussi des encodeurs et décodeurs complets [Pirsch 2003] conformes aux différents standards. Ces PIs présentent généralement une complexité de quelques milliers à quelques dizaines de milliers de portes logiques et donc une surface silicium très faible. Il suffit ensuite d'assembler ces PIs pour proposer des systèmes complets de traitement d'image. Les architectures analogiques dans le plan focal [Clapp 2004] constituent une voie de recherche prometteuse [Dubois 2008, Ginhac 2008] pour le traitement d'image rapide, mais la tendance actuelle vers la réduction de la taille des pixels ne permet pas l'utilisation de ces solutions. De plus, des composants de traitement numérique doivent toujours être implémentés pour les exploiter pleinement.

Zen [Zen 1998] a câblé en électronique numérique l'ensemble de la chaîne de traitement dans un circuit quasiment autonome associé à une mémoire d'image et un processeur de contrôle. Ce dernier, réalisé en technologie 600 nm, présente une consommation électrique de 200 mW et peut traiter des vidéos au format VGA. Ces performances en consommation se font au détriment de la flexibilité puisque seul le choix de la résolution à traiter reste autorisé.

Par la suite, *Zhou* [Zhou 2003] a réalisé un circuit de 200 kGates, soit 25 mm^2 en technologie 250 nm, qui consomme 500 mW et permet le traitement de flux vidéo de résolution VGA à une cadence de 30 images par seconde. Ce composant nécessite lui aussi une mémoire et un microcontrôleur pour fonctionner.

Nakano des laboratoires *Hitachi*, propose lui aussi une architecture dédiée [Nakano 1998] qui présente une complexité équivalente de 200 kGates pour le traitement de vidéos *NTSC/PAL/SECAM* ou d'images fixes Super eXtended Graphics Array (SXGA) (1280×1024). Là encore, une mémoire d'image doit lui être adjointe.

Ces architectures sont généralement distribuées sous forme de composants discrets ou de PIs. Ces dernières peuvent être intégrées à des systèmes globaux sur puce. Les technologies d'intégration actuelles permettent de proposer des composants plus performants tant en capacité de calcul, de résolution d'images à traiter que de fonctionnalités intégrées. C'est ainsi que les concepteurs de PIs proposent maintenant d'intégrer les algorithmes de compression et décompression conformes aux standards *H.264* ou *MPEG-4* au sein de leur circuit.

Le *v-MP2000 SD* [Kluge 2008, Videantis Inc. 2007], proposé par la société *Videantis*, consomme moins de 15 mW pour une surface silicium de 1,2 mm^2 en technologie 65 nm. Il est composé de 196 kGates associées à 84 ko de mémoire – ce qui est comparable aux architectures précédentes. En plus de la compression

et la décompression, les fonctionnalités élémentaires d'amélioration d'image, de surimpression, de conversion d'espaces colorimétriques mais aussi de débruitage par filtre bilatéral sont intégrées à l'architecture. Toutefois, si d'autres traitements doivent être supportés, comme le démosaïquage ou des algorithmes plus complexes que ceux intégrés au circuit, ils doivent impérativement être réalisés par d'autres composants. La résolution maximale supportée correspond au standard HD 720p soit 1080×720 pixels à 30 trames par seconde, ce qui correspond à 20 MPixels/s. Afin d'assurer le traitement au format HD 1080p, *Videantis* propose le *v-MP4180HDX* [Videantis Inc. 2008]. Pour afficher de telles performances, cette architecture combine quatre cœurs *v-MP2000 SD* décrits précédemment. Ce choix a un impact sur la surface silicium et la consommation électrique qui atteignent respectivement 7,9 mm^2 en technologie 65 nm et 102 mW.

2.1.2 Les architectures à base de coprocesseurs ou d'accélérateurs dédiés

Nous avons vu que les accélérateurs sont généralement associés à un processeur de contrôle et une mémoire. Afin de réduire le nombre de composants des circuits, les intégrateurs proposent des architectures de traitement d'image intégrées sur un seul composant, elles sont constituées d'un processeur ou d'un DSP associé à des accélérateurs.

Talla, de *TI*, décompose dans un article [Talla 2004] le fonctionnement du *DM310* visible en Figure 2.1a, un processeur multimédia de la firme adapté au traitement d'image. Outre les interfaces d'entrées et de sorties associées à un *ARM 9*, une mémoire de 128 ko, il est basé sur un DSP *TMS320C54x* qui possède des extensions dédiées au traitement d'image. Un premier coprocesseur est une extension vectorielle SIMD de 8 unités capables de réaliser des opérations arithmétiques et logiques plus ou moins spécialisées comme la SAD, des multiplications, des additions, ou encore des opérations basées sur l'utilisation de LUTs. Un second coprocesseur est dédié à l'opération de quantification, essentielle pour la compression *JPEG MPEG* et *H.263*. Enfin le troisième coprocesseur accélère le codage de *Huffman*, et donc les mêmes standards de compression. La consommation électrique de ce composant est annoncée à 400 mW et il peut traiter et compresser jusqu'à 9 millions de pixels par seconde, c'est à dire des flux vidéo de résolution VGA à 30 images par seconde. Cet exemple est représentatif des solutions proposées par les fabricants et notamment Texas Instruments [Texas Instruments 2006].

On voit que la flexibilité d'un composant entièrement intégré est nettement améliorée au regard des solutions entièrement câblées précédemment exposées. Toutefois, pour bénéficier des accélérateurs matériels, les algorithmes doivent se conformer aux traitements initialement proposés par le composant – par exemple les compressions JPEG, MPEG, *H.263*. Leurs consommations électriques les rendent utilisables dans le domaine de l'embarqué. De plus, l'utilisation de plusieurs PIs simultanées rend possible la parallélisation des traitements, comme c'est le cas dans les solutions Application Specific Integrated Circuits (ASICs) présentées en 2.1.1.

2.1.3 Utilisation de composants dédiés dans les SoC

Les composants de traitement d'image dédiés sont intégrés au sein d'architectures et donnent lieu à des plate-formes complètes. Celles-ci permettent de réduire notablement les coûts de fabrication en regroupant un ensemble des fonctionnalités requises au sein d'un seul et même SoC – celles pour un téléphone portable par exemple [Devices 2007]. Les fonctionnalités de traitement d'image sont une part importante du cahier des charges de ces architectures.

La plate-forme Open Multimedia Platform (OMAP) de *TI* est un exemple de ce type d'architecture [Shi 2003]. Un processeur de type *ARM cortex A8* basse consommation est associé à un DSP *TMS320C64x*, un accélérateur graphique, un processeur d'amélioration d'image « proche pixel » et un accélérateur vidéo. Ce dernier offre la capacité de calcul permettant de supporter des vidéos haute définition. Ces accélérateurs sont promus comme étant des composants dédiés afin d'obtenir des performances élevées. Les fréquences de fonctionnement de tels composants sont comprises entre 500 MHz et 1 GHz. La consommation électrique d'une telle solution est de plusieurs Watts (2 à 4 W), elle dépasse donc la contrainte fixée, et il en va de même pour la surface silicium. Néanmoins la comparaison avec d'autres solutions est difficile, puisque le circuit intègre différentes fonctionnalités dédiées dont l'utilisation permet de réduire les temps de conception des produits basés sur ces plate-formes.

D'autres fabricants de circuit proposent des composants de ce type. Citons *NVIDIA* et ses composants de la série *TEGRA* avec une surface annoncée à 144 mm^2 pour une consommation comprise entre 2,5 W et 4 W en technologie 65 nm. Le circuit intègre un processeur *ARM 11 MPCore* gérant les différentes PIs d'entrées et de sorties. Il est associé à un processeur graphique de type *GeForce*. Là aussi, un Image Signal Processor (ISP) dédié au traitement d'images de haute résolution est intégré à la plate-forme. Ce composant est combiné à un processeur vidéo supportant la compression et à la décompression.

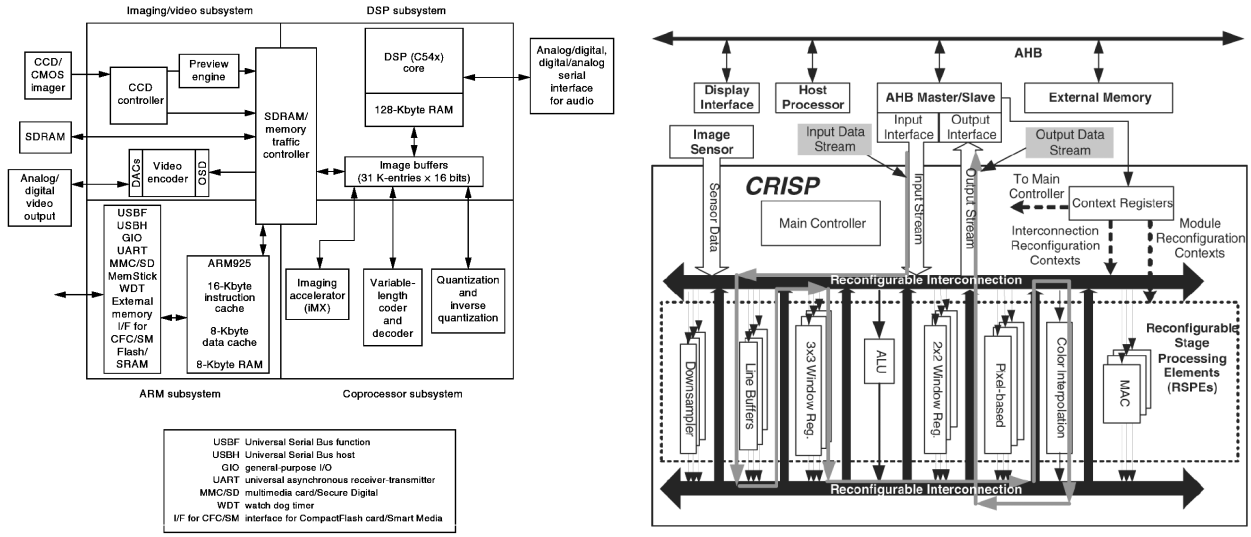
Bien que les informations sur les différents éléments qui composent ces plates-formes soient limitées en raison du marché fortement concurrentiel, une brève étude du savoir-faire de ces deux entreprises [Talla 2004, Montrym 2005, Lindholm 2008] permet de supposer que les accélérateurs dédiés au traitement d'image contiennent un ensemble d'opérateurs fonctionnant en mode SIMD associés à des PIs dédiées paramétrables.

2.1.4 Conclusion sur les architectures dédiées

De fait, les capacités de traitement des architectures dédiées sont élevées pour des consommations électriques et une surface silicium faibles. En effet, environ 200 kGates (hors éléments de mémorisation) sont généralement utilisés pour implémenter un système complet. Mais leur manque de flexibilité les rend très vite obsolètes et les composants qui les utilisent doivent être changés lorsqu'un nouvel algorithme doit être utilisé, ou encore lorsque le fabricant souhaite augmenter la résolution de ses capteurs. Ces architectures doivent le plus souvent être associées à des processeurs programmables afin de les contrôler. Ce besoin, associé au manque de flexibilité de ces solutions poussent les fabricants à proposer des systèmes intégrant toutes les fonctionnalités que nécessite un dispositif mobile tel qu'un téléphone portable, autour d'un processeur généraliste. L'utilisateur peut alors choisir d'utiliser ou non ces éléments dédiés en fonction de ses besoins et de les paramétrer en conséquence. On se place alors dans l'espace de conception des architectures reconfigurables.

2.2 Présentation des architectures reconfigurables

Les architectures reconfigurables peuvent être considérées comme une évolution des systèmes dédiés présentés précédemment. En effet, au lieu de figer dans le silicium des fonctionnalités en interconnectant des composants (transistors, portes logiques, unités arithmétiques et logiques, mémoires, etc.), la solution du reconfigurable permet de modifier les fonctionnalités implémentées en intervenant sur l'interconnexion entre ces composants après la fabrication du circuit. Pour que la reconfiguration soit efficace, il est nécessaire qu'elle soit adaptée à la fonction qu'elle doit réaliser, d'où l'importance des outils logiciels [Compton 2002] qui permettent d'exploiter les fonctionnalités matérielles proposées. Nous n'aborderons pas ici les solutions Field-Programmable Gate Array (FPGA) qui sont mal adaptées aux contraintes que nous nous sommes imposées.



(a) ORGANISATION DU *TI DM310* (b) ORGANISATION DE *CRISP*

FIG. 2.1: SCHÉMA DU PROCESSEUR (a) TRAITEMENT D'IMAGE *DM310* DE *TI*; (b) ET RECONFIGURABLE *CRISP*.

Seuls les mécanismes de reconfigurations partielles qu'ils proposent peuvent présenter un intérêt dans le cadre de cette étude [Manet 2008, Claus 2007].

Cette section présente plusieurs solutions reconfigurables pour lesquelles des approches de conception différentes ont été utilisées. Dans un premier temps, une architecture reconfigurable à gros grain est présentée. Ensuite, une architecture avec un grain de reconfiguration intermédiaire est étudiée, suivie d'une architecture à grain plus fin. Enfin, d'autres architectures reconfigurables sont brièvement abordées.

2.2.1 Coarse-Grained Reconfigurable Image Stream Processor

Parmi les architectures de traitement du signal capables de supporter des traitements HD 1080p, *CRISP* [Chen 2008a] est remarquable. Elle a été spécialement conçue pour réaliser les traitements d'amélioration d'image en aval du capteur tout en respectant les contraintes de l'embarqué. Avec 171 kGates et 74 kb de mémoire (400 kGates au total), sa surface est de 5 mm^2 en technologie 180 nm – soit près d'un millimètre carré extrapolé en technologie TSMC 65 nm – sa consommation électrique est de 218 mW à 115 MHz et elle permet de réaliser le traitement complet d'amélioration d'image sur des flux vidéo HD 1080p à 55 images par seconde.

Cette architecture est composée de six modules reconfigurables interconnectés les uns avec les autres, afin qu'ils puissent être chaînés comme on peut le voir en Figure 2.1b. Chacun d'entre-eux prend en entrée un flux de pixels, effectue le traitement, et le transmet au suivant. Le *Local Memory Module* permet de gérer les accès 2-D aux pixels sachant que seules quatre lignes sont mémorisées, il supporte donc des tailles de voisinages de 5×5, ou encore deux voisinages 2×2 (ce qui correspond à deux blocs sur la Figure 2.1b. Le *Pixel-Based Operation Module* permet de réaliser les opérations directement sur les pixels ne nécessitant pas l'accès à des données du voisinage. Un *MAC Module* assure les traitements nécessitant des opérations de multiplications et accumulations. Le *Color Interpolation Module* peut être configuré pour permettre les opérations de démosaïquage. Ce dernier communique avec le *Local Memory Module* pour obtenir les valeurs des pixels du voisinage nécessaires au démosaïquage. Une Unité Arithmétique et Logique (UAL) permet de réaliser des opérations génériques tandis qu'un dernier module permet de sous-échantillonner le flux pour sa prévisualisation.

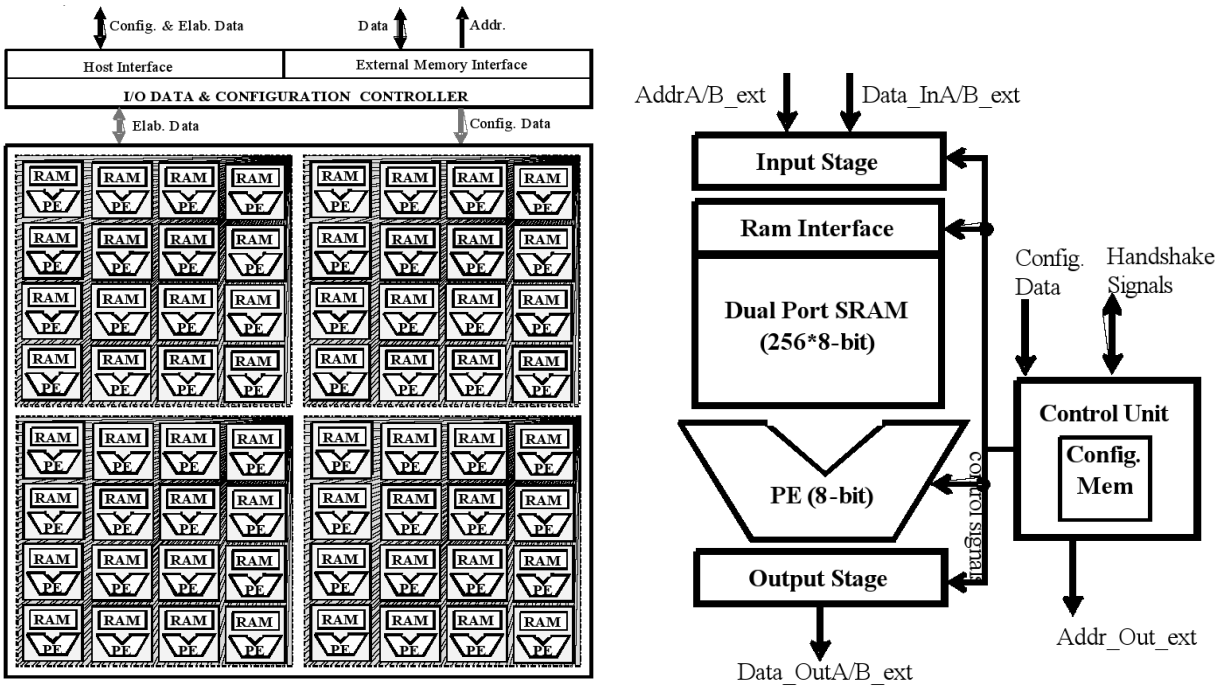
Nous retiendrons notamment le concept de module d'accès au masque de voisinage qui permet de décharger les autres modules de ces opérations, et la structure flot de données qui permet d'enchaîner les traitements et assurer un parallélisme de tâches. Les traitements que peut supporter *CRISP* restent toutefois limités à la panoplie prévue lors de la conception du circuit, essentiellement en raison de son grain élevé de reconfiguration.

2.2.2 Multimedia Oriented Reconfigurable Array

L'architecture *MORA* [Lanuzza 2007], spécialement conçue pour le multimédia, est composée d'une matrice 2×2 dont les éléments sont interconnectés par un réseau reconfigurable. Les quatre blocs sont composés de 16 cellules également reconfigurables comme on peut le voir en Figure 2.2a. Chacune d'entre-elles intègre une mémoire de 256 mots de 8 bits et une UAL de 8 bits ainsi qu'une unité de contrôle visible en Figure 2.2b. Le comportement de la cellule est déterminé par la configuration de cette unité de contrôle. La mémoire est distribuée au sein de l'ensemble des cellules, ce qui permet de limiter les communications avec l'extérieur ou entre cellules.

La surface de l'architecture *MORA* est de 0,9 mm^2 en technologie ST 90 nm, soit environ 0,6 mm^2 en technologie 65 nm. Sa capacité de traitement dépasse 300 échantillons par seconde à 1 GHz pour une DCT ou pour une conversion d'espace de couleur. La consommation électrique n'est pas renseignée mais de telles fréquences de fonctionnement induisent généralement une consommation élevée.

Une seule instance de cette architecture ne permet pas de supporter une chaîne de traitement complète. En effet, il est nécessaire de reconfigurer le circuit entre chaque traitement, ou bien d'interconnecter ensemble plusieurs instances de *MORA*, ce qui a été prévu par les concepteurs.



(a) MATRICE 2×2 DE 16 CELLULES RECONFIGURABLES (b) CELLULE RECONFIGURABLE

FIG. 2.2: ORGANISATION SCHÉMATIQUE DE L'ARCHITECTURE *MORA* (a) MATRICE 2×2 DE 16 CELLULES RECONFIGURABLES ET LEURS INTERCONNEXIONS; (b) SCHÉMATISATION D'UNE CELLULE RECONFIGURABLE.

2.2.3 Reconfigurable Instruction Cell Array

Spiral Gateway propose la matrice *RICA* [Khawam 2008] qui est une structure reconfigurable matricielle au niveau opération comme le présente la Figure 2.3. La configuration fait correspondre à chaque opération d'un programme une cellule de la matrice reconfigurable. Cette solution est particulièrement flexible puisque, associée à un processeur de calcul, elle permet d'accélérer notablement des fonctions critiques. Toutes les instructions peuvent être exécutées simultanément et sur des données différentes en vue d'obtenir une capacité de calcul élevée.

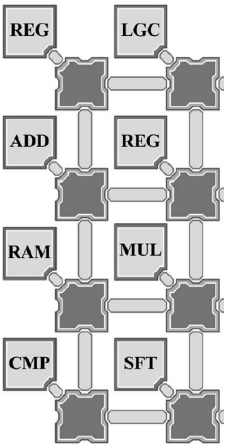


FIG. 2.3: MATRICE DE *RICA*.

2.2.4 Approche DART

L'architecture *DRIP* [Boschetti 2004] adaptée au traitement de l'image, reprend l'approche de l'architecture du processeur *DART* et tout spécifiquement au niveau du chemin de données reconfigurable dynamiquement. L'architecture *DART* [David 2003, David 2002, Pillement 2007] est prévue dès sa conception pour limiter sa consommation électrique, et l'utilisation d'opérateurs haut niveau dont l'enchaînement est reconfigurable dynamiquement permet de résoudre le problème de la flexibilité. Son fonctionnement en flux la rend adapté aux application de traitement du signal, toutefois elle est destinée au domaine des télécommunication.

2.2.5 Autres structures reconfigurables

L'architecture *MorphoSys* [Lee 2000] est composée d'un processeur Reduced Instruction Set Computer (RISC) associé à un bloc reconfigurable homogène à grain relativement fin. Ce dernier est destiné à accélérer certaines fonctions et est utilisable comme un coprocesseur. L'ensemble présente un total de 1,55 MGates, soit plus de 3 mm^2 en technologie 65 nm. Une telle surface est nettement supérieure à la précédente architecture. Sa consommation électrique, de plusieurs Watts la rend inutilisable dans le domaine de l'embarqué.

Notons aussi *ADRES* qui a fait l'objet d'une implémentation adaptée au traitement d'image par *Hartmann* [Hartmann 2007]. Cette architecture présente des consommations électriques inférieures à 30 mW, mais sa surface ramenée en technologie 65 nm dépasse plusieurs mm^2, pour moitié dédiée à la mémoire de configuration.

D'autres structures reconfigurables sont présentées dans la littérature. Elles peuvent se présenter sous la forme de coprocesseurs [Ye 2000, Stretch Inc. 2007, Hauser 1997] ou d'architectures complètes, mais peu sont adaptées aux contraintes de l'embarqué. Nous remarquons, d'une manière générale, que la limite en surface silicium est d'autant plus vite atteinte que le grain de reconfiguration est fin. Pourtant, outre la flexibilité apportée par l'utilisation d'architectures reconfigurables, le parallélisme de données peut être efficacement exploité, ce qui est tout particulièrement adapté au traitement d'image [Mérigot 1997, Biancardi 2002].

2.2.6 Conclusion sur les architectures reconfigurables

Cette brève étude des architectures reconfigurables utilisées pour le traitement d'image nous permet de constater que plus le grain de la reconfiguration est fin, plus la flexibilité de la structure sera élevée. Toutefois la surface silicium de ces architectures est inverse à la finesse du grain de reconfiguration, d'une part en raison du coût d'interconnexion, d'autre part en raison de la mémoire de reconfiguration. Dans notre cas, il est préférable de maintenir un grain de reconfiguration élevé. Ainsi la structure de calcul la plus efficace est *CRISP* mais sa flexibilité est particulièrement limitée à quelques classes d'algorithmes. Pour obtenir une flexibilité

plus importante, les structures programmables peuvent être utilisées. Ces dernières peuvent être vues comme la généralisation des architectures reconfigurables puisqu'on peut, dans ce cas, considérer que le circuit est reconfiguré à chaque instruction.

2.3 Présentation des architectures programmables

Nous avons vu dans la première section que de nombreux algorithmes de traitement d'image font l'objet d'implémentations dédiées, généralement associées à des processeurs ou des micro-contrôleurs programmables. La seconde section a présenté des structures reconfigurables adaptées au traitement d'image, plus flexibles que les solutions dédiées. Toutefois leur utilisation nécessite de déterminer avec précision le grain de reconfiguration.

Cette section présente les architectures programmables utilisées en traitement d'image. Dans la première partie de cette section, les processeurs spécifiques à une application (ASIPs) sont présentés. La seconde partie décrit les architectures de flux qui peuvent être utilisées pour des applications de traitement d'image. Ensuite, les architectures massivement parallèles SIMD font l'objet d'une troisième partie avant que nous n'introduisions les processeurs de traitement du signal. Nous terminerons par quelques données sur les processeurs embarqués généralistes et leurs ressources de calcul.

2.3.1 Les processeurs spécifiques à une application ASIP

D'une manière générale, la plupart des architectures présentées dans cet état de l'art sont des ASIPs. Cette partie présente donc l'accélération de fonctions critiques par l'extension ou l'adaptation de jeu d'instructions de processeurs. Ces opérations supplémentaires sont réalisées par des éléments dédiés ayant un accès direct aux ressources du processeur (file de registres, *bypass*, etc.). L'identification des fonctions critiques repose sur une analyse poussée des algorithmes destinés à être portés sur le composant. La littérature présente nombre de ces techniques d'analyse [Atasu 2005, Sun 2006, Chattopadhyay 2006, Jain 2001b] qui mènent à spécialiser des architectures pour qu'elles réalisent du traitement vidéo [Chouliaras 2008, Piskorski 2007].

La société *Tensilica* propose le processeur *388VDO Video Engine Architecture* [Tensilica Co. 2007] issu de son savoir-faire en terme d'analyse algorithmique pour la spécialisation de cœurs de calcul. Ce processeur met en œuvre deux cœurs *Xtensa* [Gonzalez 2000, Ezer 2000] hétérogènes, qui ont été spécialisés pour les applications de traitement vidéo et notamment de compression et de décompression. Un premier processeur VLIW traite le flux de pixels pour réaliser les opérations séquentielles et les opérations de contrôle, tandis que le second processeur, qui est un ensemble SIMD, est spécialisé pour l'accélération des opérations au niveau pixel (estimation du mouvement, transformées inverses, etc.).

Ce composant traite des flux vidéo en temps réel à résolution D1 soit 720×576 pixels. Sa surface est de 6,6 mm^2 en technologie TSMC 90 nm pour une consommation entre 30 et 60 mW – près de 3 à 4 mm^2 en technologie 65 nm. Il est par contre nécessaire de multiplier ces composants afin d'assurer plusieurs traitements simultanés, comme l'exige la chaîne de traitement d'image.

L'approche qui consiste à adapter ou étendre le jeu d'instructions d'un processeur a fait l'objet de nombreuses études [Fontaine 2008, Piskorski 2007, Sun 2005] montrant l'efficacité des ASIPs. Si pour couvrir de nombreuses applications différentes, le jeu d'instructions doit être trop augmenté, l'intérêt de ces solutions devient limité [Bauer 2008]. De plus, une trop grande spécialisation des extensions limite fortement la flexibilité

du système aux applications initialement prévues. Une analyse algorithmique avancée permet de déterminer les fonctions critiques et dimensionner efficacement une architecture.

2.3.2 Les processeurs de traitement de flux de données

Les architectures de traitement de flux sont naturellement efficaces, notamment pour le traitement d'image, puisqu'elles exploitent le parallélisme de tâches en enchaînant les différents traitements. Nombre de travaux de recherche et d'industriels ont explorés ce type d'architecture.

L'université de *Taiwan* [Tsao 2008] a développé un processeur double cœur capable d'atteindre 16 GOPs, 6,4 GFLOPs à 200 MHz pour 26 mW de consommation en technologie 90 nm. Chaque cœur est un processeur VLIW deux voies adapté au traitement d'image et notamment l'accélération 3-D.

L'architecture *Integrated Memory Array Processor (IMAP)* de *NEC* [Fujita 1995, Kyo 2001, Kyo 2005] est une architecture SIMD qui développe 100 GOPs à 100 MHz. Elle est composée de 128 processeurs VLIW à 4 voies, initialement 8 bits, puis portés à 16 bits dans sa version actuelle. Chaque processeur intègre une mémoire de quelques kilo-octets. Ces mémoires sont alimentées par un gestionnaire adapté. Cette architecture est destinée au traitement d'image embarqué au sein des véhicules pour l'assistance aux conducteurs. Seize composants peuvent être associés, multipliant d'autant sa capacité de calcul. Avec 32,7 millions de transistors, sa surface dépasserait les 15 mm^2 en technologie 65 nm, pour une consommation dépassant le Watt.

L'université de *Stanford* propose *Imagine Stream Processor* [Kapasi 2003, Dally 2001] capable d'atteindre 512 GOPs [Khailany 2008, Khailany 2007] avec la capacité de traiter des flux vidéo HD 1080p. Avec 10 W de consommation électrique, 34 millions de transistors et ses 155 mm^2 en technologie 130 nm , cette architecture portée en technologie TSMC 65 nm représenterait une surface silicium de plusieurs dizaines de millimètres carrés et de plusieurs Watts. Sa capacité de calcul est obtenue en exploitant différents niveaux de parallélisme. D'abord, l'utilisation de processeurs VLIW permet d'exploiter le parallélisme au niveau données, ensuite ces processeurs fonctionnent en mode SIMD, ce qui permet de traiter simultanément plusieurs échantillons. De plus, plusieurs groupes (clusters) de processeurs SIMD permettent de réaliser différents traitements simultanément. Nous retiendrons essentiellement que de réaliser les traitements directement dans le flux permet de limiter la mémoire du système et introduit un parallélisme temporel. Cette architecture est commercialisée par *Stream Processors Inc.* sous le nom de *STORM* [Stream Processors, Inc. 2007].

FFIESTA Les chercheurs de *Sony* proposent l'architecture *FIESTA* [Arakawa 2008] qui contient quatre unités de traitement en flux associées à une matrice de 64 processeurs SIMD chacune. Sa capacité de calcul de 512 GOPs pour une consommation électrique de 782 mW à 250 MHz lui permet de traiter des flux vidéo HD 1080p. Ce composant pourrait répondre aux contraintes de l'embarqué si sa surface en technologie 65 nm ne dépassait pas 152 mm^2 pour 20,4 MGates. La mémoire embarquée sur le circuit (> 16 Mb) couvre un tiers de la surface.

Cet état de l'art des processeurs de flux nous montre que les structures multiprocesseurs fonctionnant en mode SIMD sont largement utilisées dans ce domaine. Toutefois les surfaces de ces composants sont élevées, notamment en raison du grand nombre de processeurs qu'elles intègrent, mais aussi des éléments de mémorisation bien que le mode flux présente l'avantage de permettre la possibilité de limiter la mémoire embarquée sur le circuit.

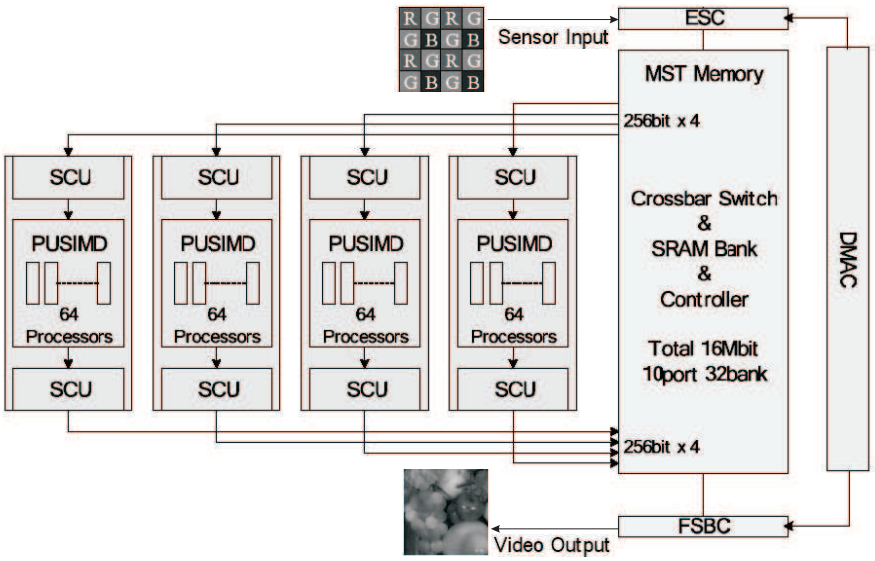


FIG. 2.4: ORGANISATION DE L'ARCHITECTURE *FIESTA*.

2.3.3 Les accélérateurs graphiques GPU

Les accélérateurs graphiques [Montrym 2005] peuvent s'apparenter à des processeurs de flux programmables et être utilisés comme tels [Venkatasubramanian 2003], ou catégorisés dans les ASIPs. L'essor de leur développement est lié au rendu 3-D temps réel, mais ces architectures de plus en plus flexibles et programmables, trouvent des applications dans le traitement vidéo en temps réel de par leurs importantes capacités de calcul. La forte demande dans le domaine des dispositifs mobiles rend leur étude incontournable, avec plus du tiers du marché du GPU et 600 millions d'unités vendues [Montrym 2005] en 2005.

La tendance actuelle de *NVIDIA* et de sa nouvelle architecture *Tesla* [Lindholm 2008] est d'utiliser plusieurs ensembles de processeurs SIMD, comme dans les précédentes générations [Montrym 2005], avec l'ajout de fonctionnalités multi-threads. Cette architecture SIMD, schématisée en Figure 2.5, est commercialisée au sein du composant *NVIDIA* GeForce 8800 qui compte 681 millions de transistors et représente une surface 470 mm^2 en technologie TSMC 90 nm. Ce processeur graphique développe des performances record de 576 Gflops et près de 500 GOPs à 1,5 GHz pour une consommation de 150 W.

Woo propose un accélérateur graphique intégrant des fonctionnalités de traitement vidéo pour le domaine de l'embarqué [Woo 2008, Woo 2007] puisque sa consommation de 152 mW est acceptable (en 130 nm). Toutefois, ses 18,6 millions de transistors (la surface équivalente en technologie 65 nm est de l'ordre de la dizaine de millimètres carrés) dépassent de plusieurs ordres de grandeur la contrainte en surface. Ces performances sont obtenues grâce à l'utilisation d'éléments dédiés associés à un cœur de calcul SIMD.

D'autres accélérateurs graphiques conçus par différents instituts de recherche sont proposés dans la littérature [Chen 2009, Yoo 2007]. La plupart d'entre eux s'appuient sur des UAL SIMD pour atteindre des performances permettant d'assurer le traitement 3-D en temps réel. Nous citerons *Mali* [Corp. 2009], le processeur graphique développé par *ARM*. Il est annoncé par le constructeur comme étant le plus petit GPU disponible – *Mali-55* présente une surface de 1 mm^2 pour 100 MPixels/s.

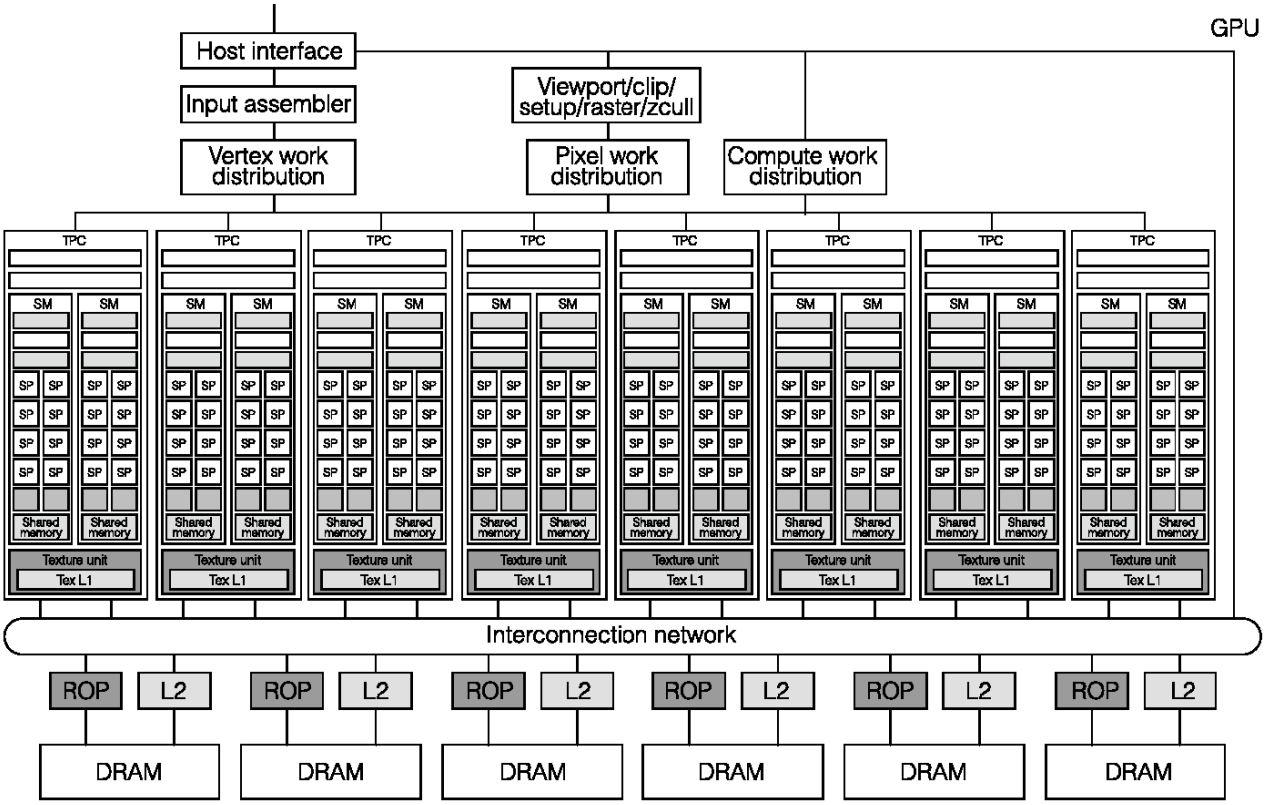


FIG. 2.5: ORGANISATION DE L'ARCHITECTURE *NVIDIA TESLA*.

D'une manière générale, nous retiendrons que ces architectures s'appuient sur des structures SIMD, comme c'est le cas pour les processeurs de flux précédemment vus. Les implémentations de ces circuits dédiés aux applications embarquées respectent les contraintes de consommation et de surface silicium mais ils sont essentiellement conçus pour l'accélération 3-D. La sous-section suivante est consacrée à l'étude des architectures programmables SIMD qui semblent maintenant incontournables afin d'assurer flexibilité et capacité de calcul.

2.3.4 Les architectures parallèles SIMD

Les architectures SIMD traitent plusieurs données simultanément, en appliquant les mêmes instructions sur des données différentes. Pour cela, les éléments de calcul doivent être dupliqués autant de fois qu'il y a de données à traiter en parallèle. Par contre, les ressources de contrôle et la mémoire programme ne sont pas dupliquées.

SYMPATIX [Collette 1992a], qui a par la suite donné lieu à *Symphonie* [Collette 1997], est une architecture SIMD de 1024 processeurs élémentaires intégrant des accélérateurs pour les calculs à virgule flottantes. Ces processeurs sont interconnectés entre-eux par un réseau de communication à jetons spécifiquement conçu [Collette 1992b, Letellier 1993]. Cette architecture développée dans notre laboratoire est notamment exploitée pour des applications de vision infrarouge en temps réel dans les avions de chasse supersoniques [Collette 2001].

L'architecture *iVisual* [Cheng 2008], proposée par l'université de *Taipei* est capable de développer 76,9 GOPs pour une consommation électrique de 374 mW grâce à une fréquence de fonctionnement de seulement 50 MHz. Sa surface en technologie 180 nm est de 70 mm^2, ce qui correspond à environ 10 mm^2 en technologie 65 nm.

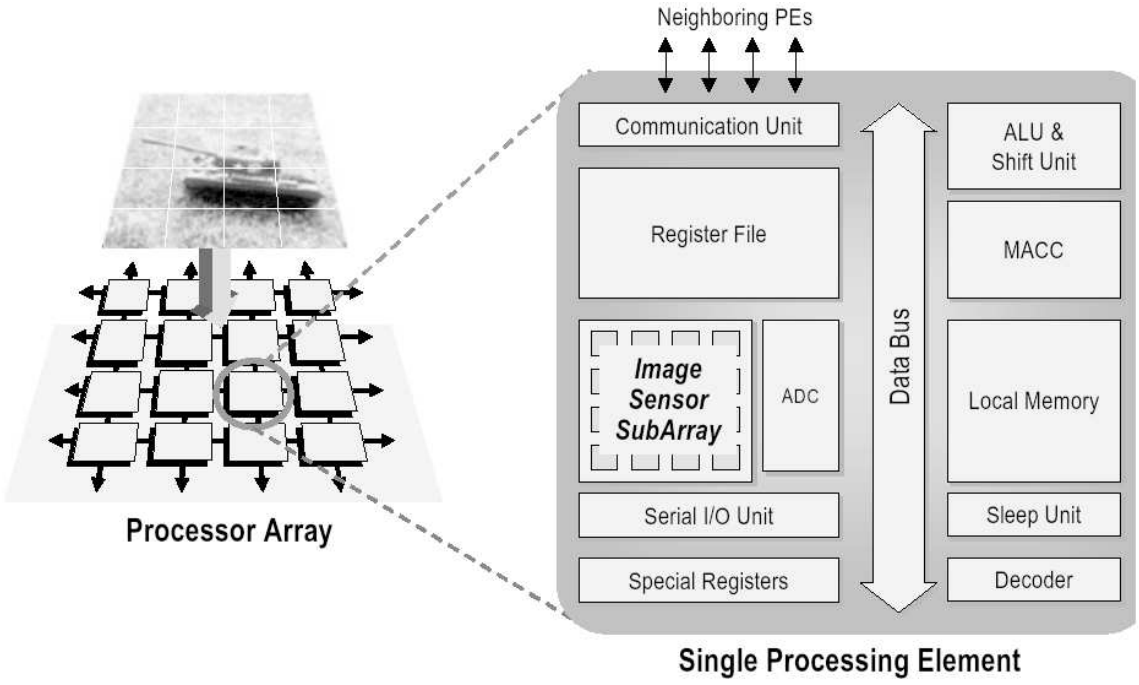


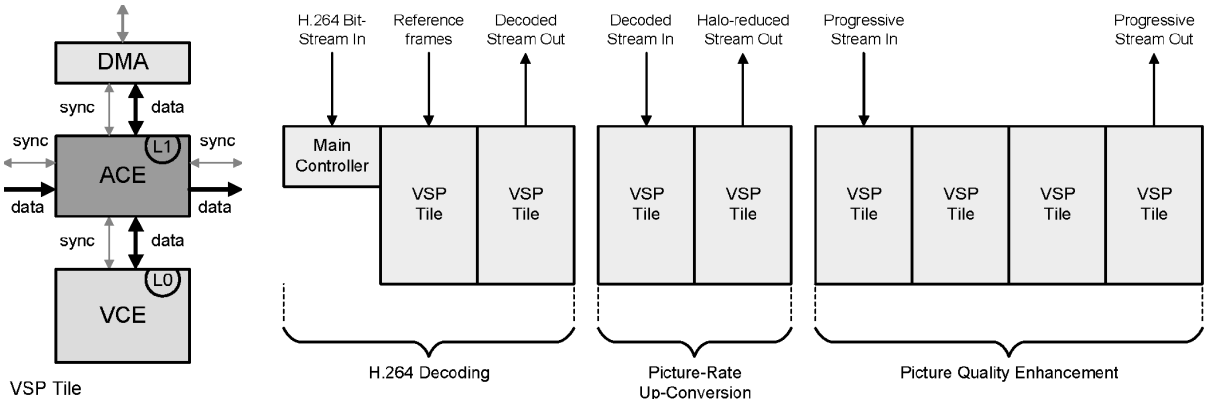
FIG. 2.6: REPRÉSENTATION SCHÉMATIQUE DE L'ORGANISATION DE *SIMPil* SUR LE PLAN FOCAL.

Cette architecture est construite autour d'un bloc de 128 processeurs élémentaires et d'une hiérarchie mémoire. Un processeur de décision supplémentaire de type MIPS est ajouté au circuit rendant l'architecture particulièrement adaptée aux opérations de reconnaissance.

L'architecture *VIRAM* [Kozyrakis 2002a, Kozyrakis 1999] est composée d'un processeur *MIPS* 64 bits associé à un coprocesseur vectoriel de 4×64 bits interconnectés à 8 bancs mémoire qui représentent 13 Mo de données. L'ensemble est interconnecté par un réseau crossbar. Avec 7,5 millions de transistors (hors mémoire), il consomme 2 Watts en technologie 180 nm pour une capacité de calcul de 1.6 Gflop/s en 32 bits. Même si elle n'est pas adaptée aux contraintes de l'embarqué, cette architecture a démontré ses performances par rapport aux architectures superscalaires et VLIW [Kozyrakis 2002b] et se place en concurrence avec d'autres architectures parallèles comme *Imagine* [Chatterji 2003].

Cat et *Gentile* ont proposés l'architecture *SIMPil* [Cat 1996] destinée à être intégrée directement dans le plan focal auprès des pixels comme le schématise la Figure 2.6. Cette organisation qui permet de réaliser directement les opérations de la chaîne de traitements et d'amélioration sur les pixels [Gentile 2005] : un processeur est associé à un groupe de 4×4 pixels directement au niveau du capteur. Si cette architecture peut atteindre le TOPs, elle reste difficilement envisageable au sein d'un dispositif à bas coût. En effet, la version de 64×64 pour traiter des images 256×256 présente une surface rédhibitoire de 150 mm^2 en technologie 350 nm [Gentile 2004] – près d'une dizaine de millimètres carrés en technologie 65 nm. Les nouvelles technologies d'intégration 3-D permettent de revisiter ce type d'architecture.

Les chercheurs des laboratoires de *Philips NXP* ont proposé l'architecture *Xetal* [Kleihorst 2001]. Elle est composée de 320 processeurs de 16 bits avec des performances pouvant atteindre 107 GOPs pour une consommation électrique de 600 mW [Abbo 2008]. La surface de cette architecture en technologie 90 nm est de 74 mm^2



(a) UNE TUILE *VSP* (b) ENCHAÎNEMENT DE PLUSIEURS TUILES *VSP*

FIG. 2.7: ORGANISATION DE L'ARCHITECTURE *Hive'flex* DE *Silicon'Hive* (a) UNE TUILE *VSP* QUI CONTIENT LES TROIS MODULES DMA, *ACE* ET *VCE* ; (b) PLUSIEURS TUILES ENCHAÎNÉES POUR RÉALISER UNE APPLICATION DE TRAITEMENT D'IMAGE.

– soit plus de 30 mm^2 en 65 nm. Ainsi, même si sa surface silicium ne permet pas d'envisager son utilisation en téléphonie mobile, sa capacité de calcul, sa consommation et son modèle architectural la rendent particulièrement adaptée au traitement d'image proche pixel.

L'architecture *Hive'flex* de *Silicon'Hive* [Beric 2006, Pinto 2006] est conçue pour le marché de la vidéo domestique. Il s'agit de tuiles de calcul que le fabricant dimensionne et organise avant la fabrication du circuit. Elles s'articulent autour d'un bus de données. Comme le présente la Figure 2.7a, ces tuiles sont constituées, en plus d'une unité de contrôle, de trois modules: un module d'accès à la donnée Direct Memory Access (DMA) et deux modules SIMD. Le premier traite le niveau applicatif (*ACE*) et contient la région d'intérêt à traiter. Il permet aussi de commander le second module qui contient les pixels sur lesquels opérer et réalise les calculs à proprement parler (*VCE*). Ces tuiles peuvent ensuite être interconnectées les unes autres pour réaliser une application comme le montre la Figure 2.7b. Un composant contenant seize de ces tuiles présente une surface de 8,1 mm^2 en technologie 90 nm – soit approximativement 5 *mm*^{2} en technologie 65 nm – pour une performance 16 bits équivalente annoncée à 520 GOPs à 200 MHz). La consommation annoncée est de quelques centaines de mW. Cette architecture permet donc d'assurer le traitement vidéo HD 1080p [Silicon Hive 2008a, Silicon Hive 2008b].

Les architectures de traitement parallèle SIMD permettent d'obtenir des ressources de calcul autorisant le traitement de type vidéo HD 1080p pour une consommation électrique de quelques centaines de milliWatts à quelques Watts. Leur surface silicium reste souvent élevée en raison de l'utilisation de mémoires distribuées. Ces architectures possèdent souvent une hiérarchie mémoire afin d'éviter des contentions lors des accès aux données et limiter la communication entre les différents éléments. De plus, il n'est pas rare que des processeurs VLIW soient utilisés en tant qu'éléments de calcul.

2.3.5 Les processeurs de traitement du signal DSP

Le marché des processeurs de traitement du signal est particulièrement concurrentiel. Afin d'assurer l'utilisation d'un large spectre d'application et de limiter les coûts d'intégration en réutilisant leurs architectures, les chercheurs conçoivent des processeurs suffisamment flexibles pour supporter aussi bien du traitement d'image que des applications audio ou de télécommunication. Si chaque fondeur propose plusieurs DSP à son catalogue,

nous avons choisi de présenter quelques composants représentatifs des performances qui peuvent être atteintes.

Les DSP *C6xx* de *TI* implémentent l'architecture *Velocity* [Agarwala 2002]. Différents modes d'accès aux données assurés par des dispositifs DMA permettent de réduire le coût d'accès. Afin d'exploiter le parallélisme au niveau instruction, comme le *C62x*, le *C64x* est un processeur VLIW 8 voies, réparties sur deux chemins de données séparés comme le montre la Figure 2.8. Elles incluent six UAL et deux unités de chargement et de stockage des données. De plus, des instructions SIMD sont intégrées, lui permettant de manipuler des vecteurs de données. A 600 MHz, le cœur consomme 718 mW pour une capacité de 4,8 GMACs 8 bits, à 300 MHz, sa consommation est de 200 mW. Le C64x est composé de plus de 64 millions de transistors, soit une surface de plusieurs millimètres carrés en technologie 65 nm.

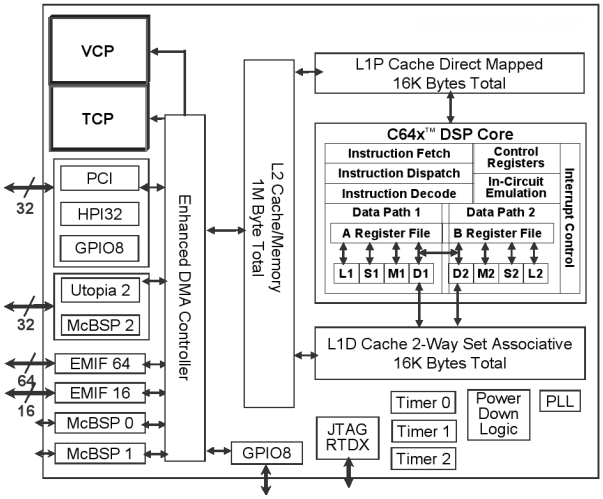


FIG. 2.8: ARCHITECTURE SCHÉMATISÉE DU DSP *TI C64x*.

L'architecture *SH-5* développée par *Hitachi* [Biswas 2000] permet d'assurer une capacité de calcul de 360 MIPs à 200 MHz en ne consommant que 80 mW [Kamae 2005]. Cette capacité de calcul est, là encore, obtenue grâce à des extensions SIMD intégrées au processeur. Toutefois la surface de ce composant est de plusieurs millimètres carrés en technologie 65 nm.

Philips a aussi développé DSP *Trimedia* [Eijndhoven 1999]. Il s'agit d'un processeur VLIW de 5 voies qui est associé à une unité SIMD et une DMA. Sa consommation varie en fonction des implémentations, mais sa capacité de calcul n'est pas suffisante pour supporter l'ensemble de la chaîne de traitement vidéo HD 1080p.

Fort du constat que les DSP VLIW sont souvent associés à un processeur généraliste, et que le nombre de voies a un impact majeur sur la surface de la file de registres, l'université de *Taiwan* propose *Pica* [Lin 2005], un DSP contenant un processeur RISC associé à un processeur VLIW deux voies. Le cœur de ce processeur représente 343 kGates (643 kGates mémoire incluse) et consomme 380 mW à 208 MHz. En technologie 65 nm, sa surface serait de l'ordre du millimètre carré pour une capacité de calcul annoncée comparable à un *TI C64x*.

Remarquons que ces DSP sont généralement des processeurs VLIW permettant d'exploiter au mieux le parallélisme d'instruction, certains intègrent aussi des extensions SIMD. L'utilisation des dernières technologies basse consommation assure une consommation correspondant aux contraintes que nous nous sommes fixées, bien que ce ne soit pas le cas de la surface silicium. De plus, un seul de ces DSP n'est pas suffisant, aussi des co-processeurs ou une architecture combinant plusieurs de ces processeurs doit être mise en œuvre [Thevenin 2006, Brost 2006] pour assurer la capacité de calcul nécessaire au traitement vidéo embarqué. Dans ce cas, la surface silicum totale devient rédhibitoire.

2.3.6 Processeurs généralistes embarqués

Le maximum de flexibilité est assuré par les processeurs généralistes que les concepteurs s'efforcent de rendre le moins consommant possible pour des surfaces toujours plus réduites, notamment grâce aux nou-

velles technologies d'intégration basse consommation. Les plus connus et les plus utilisés sont les processeurs *ARM* [Ryzhyk 2006, Goodacre 2005, Steve Furber 2000], *MIPS* [MIPS Co. 2005] ainsi que le *PowerPC* [IBM 2003] d'*IBM* et *Motorola* et le *Cell*. De plus, nombre d'entre-eux intègrent des extensions DSP ou SIMD [Glossner 2000] destinées à accélérer les calculs de traitement du signal. Malgré tout, leur consommation dépasse très souvent le demi Watt, pour une surface silicium de plusieurs millimètres carrés. De plus, leur capacité de calcul reste insuffisante pour assurer le traitement vidéo HD en temps réel. Ainsi, comme dans le cas des DSP, plusieurs instances de ces composants doivent être associés. Ils sont aujourd'hui utilisés pour les applicatifs de haut niveau, conjointement à des extensions dédiées qui assurent les traitements les plus coûteux.

Citons l'initiative de *Cortus* et son processeur 32 bits *APS3* [Cortus 2008] qui peut intégrer un multiplieur pour 8 à 12 kGates et une surface de 0,04 mm^2 en technologie TSMC 65 nm. Même si sa capacité de calcul reste limitée, sa surface permet d'envisager l'utilisation conjointe de plusieurs cœurs de ce type.

Conclusion

La première section a montré que de nombreux algorithmes de traitement d'image faisaient l'objet d'implémentations dédiées. Celles-ci sont généralement associées à des processeurs ou des micro-contrôleurs au sein de plate-formes complètes. Les contraintes de surface et de consommation sont respectées pour une capacité de calcul permettant le traitement vidéo HD 1080p. Il est toutefois impossible de modifier les algorithmes câblés sur ces composants, à moins de les remplacer.

Les architectures reconfigurables sont proposées afin d'introduire de la flexibilité. Lorsque le grain de reconfiguration est élevé comme pour *CRISP*, les performances sont proches des ASICs, mais la flexibilité reste limitée aux cas prévus par le concepteur. Au contraire, les architectures reconfigurables à grain fin permettent d'obtenir une flexibilité élevée, par contre, le surcoût en surface rend leur utilisation impossible dans le cas de l'embarqué, notre domaine d'intérêt. Grâce à une accélération des fonctions critiques identifiées par analyse algorithmique, les ASIPs permettent de répondre aux contraintes de surface et de consommation imposées, toutefois leur flexibilité reste limitée aux applications initialement prévues. Il est donc difficile ou impossible de faire évoluer les traitements une fois le circuit conçu.

Les processeurs programmables se déclinent en plusieurs familles. Les architectures flux et les architectures SIMD sont capables d'assurer des capacités de calcul élevées avec des consommations électriques faibles. Certaines de ces architectures intègrent des processeurs VLIW qui permettent d'exploiter le parallélisme au niveau instruction. Les structures SIMD sont aussi très largement répandues au sein des processeurs graphiques qui font partie des architectures de calcul les plus performantes aujourd'hui. Les processeurs VLIW sont largement utilisés au sein des DSP actuels intégrant des fonctionnalités SIMD. Comme c'est le cas pour les processeurs généralistes embarqués, leur capacité de calcul limitée nécessite qu'ils soient associés à des coprocesseurs dédiés ou que plusieurs d'entre-eux soient utilisés pour assurer les ressources nécessaires au traitement vidéo HD 1080p.

D'une manière générale, les architectures SIMD en mode flux présentent un rapport performance, surface et consommation au dessus des architectures programmables usuelles et de traitement du signal. On remarque systématiquement l'utilisation d'un module dédié à l'accès aux données, tandis que certaines structures sont conçues pour former un enchaînement exploitant le parallélisme inhérent à la chaîne d'amélioration d'image. L'utilisation des différentes formes de parallélisme mis à disposition par le traitement d'image est donc incontournable [Mérigot 2008].

CHAPITRE 3

Détermination du modèle architectural

Où est présentée l'analyse algorithmique conduisant aux choix architecturaux.
Où est proposé un modèle d'architecture d'après les résultats de cette analyse.

Introduction

LE CHAPITRE PRÉCÉDENT a montré que de nombreuses architectures se côtoient dans le domaine du traitement d'image. L'analyse algorithmique est une étape incontournable qui permet de proposer une structure adaptée, ceci est particulièrement vrai pour les ASIPs. D'une manière générale, les solutions programmables offrent le niveau de flexibilité que nous attendons. Celles fonctionnant en mode SIMD et en flux de données présentent des capacités de calcul élevées mais avec une surface silicium encore importante. Nombre de ces architectures utilisent des processeurs élémentaires VLIW pour exploiter le parallélisme d'instructions des programmes.

Ce chapitre présente la méthode mise en œuvre pour déterminer les choix architecturaux. La première section présente la méthodologie de l'analyse des algorithmes. Celle-ci détermine les modes d'accès aux données et montre les différents niveaux de parallélisme exploitables. Les opérateurs essentiels au traitement d'image sont aussi recherchés. La seconde partie de ce chapitre présente le modèle architectural qui découle de cette analyse. Ce modèle tient compte des différentes architectures rencontrées dans l'état de l'art afin de proposer une structure originale et adaptée.

3.1 Analyse des algorithmes

La première partie de cette section présente la méthode que nous avons définie pour déterminer les ressources à intégrer au sein de l'architecture de calcul que nous nous proposons de concevoir. Cette méthode se base sur une étude des algorithmes représentatifs des chaînes d'acquisition et de reconstruction d'image qui a été introduite dans le premier chapitre.

Outre les approches de conception des ASIPs abordées dans l'état de l'art, différentes techniques permettent l'estimation des ressources de calcul des algorithmes : les simulateurs de processeurs au niveau instruction permettent d'obtenir des résultats précis au cycle processeur près. Des outils tels que *Valgrind* permettent de profiler l'exécution d'un programme et d'en compter les appels réalisés aux instructions ou les ressources mémoires [Nethercote 2007] utilisées. Toutefois ces solutions contraignent le choix de l'architecture puisqu'elles sont basées sur des modèles d'architectures déjà existantes. L'utilisation d'outils pour l'exploration de l'espace de conception des ASIPs, des processeurs reconfigurables [Mucci 2005] et des processeurs à jeu d'instructions extensibles de [Goodwin 2007] permettent de s'affranchir en partie aux limitations qu'imposent un jeu d'instruction donné.

Pour assurer l'indépendance des résultats par rapport au type de processeur et ne pas contraindre les choix architecturaux, nous avons conçu la suite d'outils MAsS. Ses outils permet d'extraire des statistiques à partir de l'exécution d'algorithmes sur un jeu de données réel. Elle permet aussi de construire le graphe d'enchaînement

des opérations relatif à une exécution donnée. L'analyse de ce graphe détermine ensuite les opérateurs qui doivent être supportés par l'architecture ainsi que les niveaux de parallélisme exploitables.

La seconde partie de cette section présente le modèle architectural auquel l'ensemble des résultats de l'étude algorithmique nous conduit. Les choix réalisés doivent nous permettre de parvenir à la capacité de calcul recherchée tout en limitant la surface silicium et la consommation aux niveaux initialement fixés. Rappelons que ces niveaux sont respectivement de quelques millimètres carrés et quelques centaines de milliwatts.

3.1.1 Méthode d'analyse des algorithmes

Méthodologie générale La méthode permettant d'obtenir précisément les opérations réalisées par l'exécution d'un algorithme donné repose sur l'analyse dynamique du graphe d'exécution. Ce graphe est généré à partir de son exécution, dans notre cas, il correspond à l'enchaînement des opérations exécutées. Le jeu de données de référence est composé des images de la base de données Kodak [Kodak Eastman Co. 1995], ainsi que d'images issues de prises de vues du monde réel dans différentes conditions pour s'assurer des résultats réalistes. Le profil dynamique est réalisé avec l'aide de la chaîne d'outils *MAsS* que nous avons conçue et développée pour construire et analyser le graphe d'exécution à partir de l'exécution réelle d'un programme mais aussi des informations sur les ressources utilisées (opérations, accès mémoire, registres etc.).

Description de la chaîne d'outils *MAsS*

Pour profiler l'exécution d'un programme, instruction après instruction, chacune d'entre elles est décrite et un comportement lui est associé. Cela revient à obtenir un langage au niveau instructions, autrement dit un assembleur à partir duquel nous décrivons les algorithmes à étudier. Nous avons choisi de définir un jeu d'instructions élémentaires réduit, de type RISC. Le jeu d'instructions choisi peut aussi bien être celui d'un processeur existant – ce qui permet d'utiliser un compilateur – qu'un jeu d'instructions entièrement défini.

Afin de ne pas imposer de contrainte liée à un modèle architectural existant, nous avons choisi d'implémenter notre propre langage assembleur. De cette manière, il est aussi possible d'évaluer l'impact de l'utilisation d'extensions au jeu d'instructions initial sur l'exécution d'un programme. Il s'agit par exemple de l'utilisation de gestionnaires d'adresses, d'extensions vectorielles, ou encore d'intervenir sur la dynamique des opérations (8 bits, 16 bits, 64 bits etc.). Le code à analyser peut être inséré en ligne dans un programme ou une fonction pré-existante `C/C++`, de manière à cibler les fonctions à analyser.

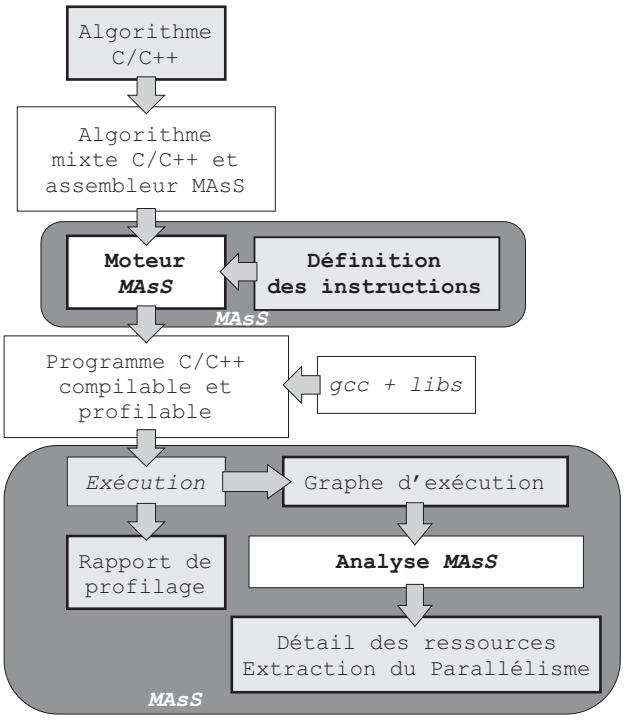


FIG. 3.1: MÉTHODE DE GÉNÉRATION DU GRAPHE D'EXÉCUTION ET D'OBTENTION DES RÉSULTATS DE PROFILAGE À PARTIR D'UN ALGORITHME DE *MAsS*.

L'algorithme ainsi décrit est intégré puis ensuite analysé par *MAsS* qui génère un programme `C` compilable. Comme c'est le cas pour *gprof* [McKusick 1982], l'exécution du programme compilé reste fonctionnellement équivalente à la version originale. Il intègre un ensemble de primitives qui permet de collecter des statistiques sur son exécution et de générer le graphe d'exécution. La Figure 3.1 présente le flot qui permet d'obtenir les résultats complets du profilage d'un programme.

L'analyse du graphe d'exécution est réalisée par un dernier outil de *MAsS*. Il permet d'extraire des sous-graphes, le parallélisme au niveau instructions, ou générer un graphe pour l'affichage. La chaîne d'outils *MAsS* est écrite en langage *Practical Extraction and Report Language (Perl)* [Wall 200] et en langage `C`. La mise en forme des graphes est assurée par la suite d'outils *Graphviz* [Gansner 1999].

« Rôle » des instructions Afin d'obtenir un profil fiable, nous avons choisi de marquer les instructions en fonction de leur tâche assignée dans le programme. Pour cela, trois « rôles » ont été définis. Le premier est destiné à identifier les instructions associées aux accès aux données, c'est à dire le calcul des adresses mémoire et le chargement ou l'écriture de valeurs. Le second « rôle » permet de marquer les instructions associées au contrôle du programme. Il s'agit notamment de celles permettant la gestion des boucles traitant le parcours de l'image et la gestion des indices. Enfin, le troisième groupe marque les instructions qui contribuent directement au résultat de l'algorithme.

3.1.1.1 Définition de la méthode d'analyse et d'obtention des résultats

A partir du programme décrit à l'aide du langage assembleur que nous avons défini, associé au modèle comportemental des instructions utilisées, un programme `C` compilable est généré. Il intègre les primitives permettant de collecter des statistiques sur les instructions exécutées et leur enchaînement. Celles-ci vont du nombre d'accès réalisés à la file de registres, aux opérateurs utilisés en passant par les erreurs de dépassement de capacité reportées.

Dynamique des calculs La dynamique des opérateurs de calcul est obtenue expérimentalement. Pour cela, nous faisons varier leur taille jusqu'à ce qu'un dépassement de capacité soit reporté, ou jusqu'à ce que l'erreur obtenue sur les images traitées dépasse un seuil donné. Cette erreur est calculée en utilisant la somme de la différence pixel à pixel au carré de l'image de référence avec l'image traitée. Pour faire varier la taille des opérateurs, il suffit de paramétrer le fichier de configuration de *MAsS*.

Construction du graphe d'exécution Le graphe d'exécution est construit à partir des différentes ressources auxquelles l'exécution du programme a fait appel. Il correspond à l'enchaînement de toutes les opérations et des résultats de calculs intermédiaires. Pour cela, chaque ressource à laquelle le programme fait appel est associée à un nœud du graphe d'exécution (registre, opérateur, constante etc.). Le « rôle » de chaque nœud est renseigné afin de pouvoir identifier ultérieurement la nature de l'utilisation des ressources associées.

Pour expliciter au mieux le parallélisme du programme et pour ne pas dépendre des ressources définies (nombre de registres, nombre d'opérateurs etc.), le graphe d'exécution est construit en renommant systématiquement les registres et les opérateurs. La méthode employée pour renommer ces différents éléments est illustrée dans l'exemple de la Figure 3.2. Cet exemple présente une fonction linéaire simple $y = a \times x + b$ pour laquelle les registres et les opérateurs sont suffixés du numéro correspondant à leur appel. La lecture du graphe permet de visualiser les données manipulées ainsi que les opérateurs en jeu, avec le parallélisme exploitable.

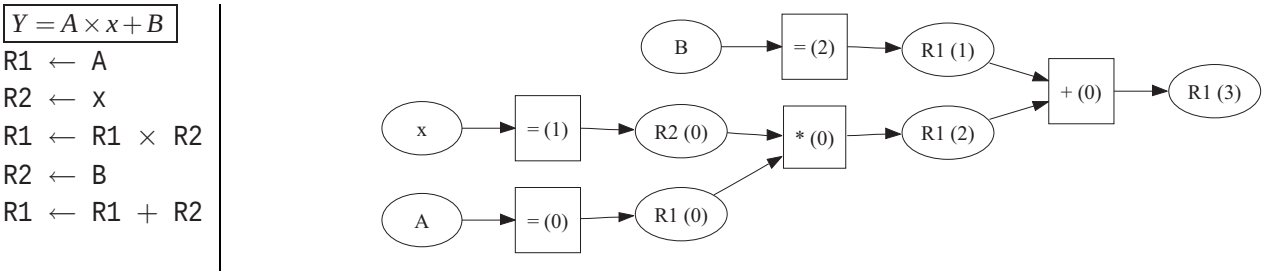


FIG. 3.2: TRADUCTION DE LA FONCTION $y = a \times x + b$ EN OPÉRATIONS ÉLÉMENTAIRES PUIS EN GRAPHE D'EXÉCUTION.

Conclusion Les fonctions à identifier doivent être décrites dans un langage prédéfini au niveau instructions. *MAsS* permet ensuite de générer une version compilable qui intègre les primitives permettant d'obtenir des statistiques précises sur les appels des instructions, l'accès aux registres et les données manipulées, ainsi que le graphe d'exécution complet du programme. Afin d'obtenir des résultats réalistes, le programme doit être exécuter sur différents jeux de données réelles. Chacune de ces exécutions fait intervenir plusieurs millions ou milliards d'accès aux opérateurs. Or, à chacun de ces appels correspond un nœud du graphe d'exécution. Il est donc essentiel de définir des stratégies permettant de réaliser une analyse efficace de ces graphes particulièrement volumineux.

3.1.1.2 Définition de la méthode d'analyse du graphe

Le graphe d'exécution est construit à partir des ressources auxquelles le programme a fait appel durant son exécution. Par conséquent, il peut contenir plusieurs milliards d'occurrences. Ce graphe ainsi constitué contient intrinsèquement les différentes formes de parallélisme ainsi que le décompte et l'enchaînement des différents opérateurs. Afin d'extraire efficacement ces informations, différentes méthodes d'analyse doivent être mises en œuvre.

La première étape consiste à segmenter le graphe d'exécution en extrayant des sous-graphes. Pour cela, il est par exemple possible de se baser sur des attributs associés aux opérations, en les regroupant en fonction de leurs « rôles » (accès aux données, contrôle, calcul du résultat). Pour chacun des sous-graphes obtenus, nous recherchons les suites d'opérations similaires, ce qui conduit à étendre le jeu d'instructions initialement défini au niveau de *MAsS*. Dans ce cas, le programme doit à être à nouveau décrit en tenant compte de ces extensions, puis analysé à nouveau pour déterminer l'impact de son utilisation sur l'exécution du programme (nombre d'opérations et de registres nécessaires, etc.). Cet impact est obtenu par comparaison des décomptes d'opérations (un poids peut être associé aux instructions en fonction de leur éventuel coût architectural). Pour cela, il suffit de décompter les nœuds qui correspondent à des opérations, il s'agit des nœuds de forme carrée dans la Figure 3.2. Enfin, ce graphe met en évidence différentes formes de parallélisme.

Extraction des formes de parallélisme Dans un premier temps, nous cherchons à identifier le parallélisme de tâches. Il s'agit des tâches qui peuvent être réalisées simultanément à d'autres, par exemple en les enchaînant. Généralement, une partie de ce parallélisme de tâches a été identifiée manuellement lors de l'écriture du programme. Les « rôles » attribués aux opérations (accès aux données, contrôle, calcul du résultat) permettent d'identifier différentes tâches qui peuvent être exécutées en parallèle.

Dans un second temps, nous cherchons à identifier le parallélisme spatial qui est clairement mis en évidence lorsque le graphe est décomposé en sous-graphes comme nous le verrons par la suite. Cette forme de parallélisme correspond aux traitements réalisables en mode SIMD; elle est particulièrement visible sur le graphe d'exécution lorsque le calcul réalisé sur le pixel courant ne dépend pas du résultat du calcul sur le pixel précédent.

Enfin, le parallélisme au niveau instructions nécessite une étude spécifique, qui peut être globale, ou sur chacun des sous-graphes à étudier. Il peut être obtenu grâce au décompte du nombre d'opérations pouvant être exécutées simultanément. On voit par exemple, en lisant la Figure 3.2 de gauche à droite, qu'au maximum trois opérations peuvent être exécutées simultanément. Pour obtenir précisément ce parallélisme, les nœuds sources du graphe d'exécution sont recherchés et décomptés. Ensuite, ils sont supprimés, ce qui a pour effet de produire un nouveau graphe ayant lui même une ou plusieurs sources. Dès lors, l'opération est répétée itérativement jusqu'à ce que le graphe ne comporte plus aucun nœud. Le parallélisme moyen au niveau instructions correspond à la moyenne de ces décomptes.

3.1.2 Résultats de l'analyse des algorithmes

Afin de définir une architecture apte à supporter ces algorithmes, il est nécessaire de déterminer précisément le format des images à manipuler, les besoins en ressources de calcul ainsi que les opérations utilisées. Comme nous avons pu le voir en analysant l'état de l'art des architectures, exploiter les différentes formes de parallélisme est crucial pour concevoir une architecture de calcul efficace.

La section suivante présente les résultats obtenus par l'analyse algorithmique en employant la méthode décrite dans la précédente partie de cette section. D'abord, les besoins en ressources de calcul sont exposés pour les différents algorithmes, suivis dans une seconde partie par les types et les formats des données manipulés. Enfin, la dernière partie présente les résultats qui découlent de notre analyse des différentes formes de parallélisme.

3.1.2.1 Détermination des besoins en ressources de calcul

A partir de la méthodologie mise en place, il est possible de déterminer les ressources de calcul nécessaires à l'exécution des algorithmes d'amélioration d'image. Ces ressources sont exprimées en opérations par pixel. La mesure a été effectuée sur des jeux de données afférents à des traitements vidéo en HD 720p à des résolutions 1080×720 et en HD 1080p 1920×1080 à 25 images par seconde. Ces ressources correspondent aux opérations associées aux instructions préalablement définies dans `MAsS` à partir desquelles les algorithmes ont été décrits.

Ressources de calcul nécessaires à l'exécution des algorithmes La Table 3.1 présente le décompte des ressources nécessaires à l'exécution de chacun des algorithmes d'une selection représentative des traitements de la chaîne d'amélioration vidéo. La plupart des traitements nécessitent quelques milliards d'opérations par seconde (GOPs), équivalent à quelques dizaines d'opérations par pixel en vidéo HD 1080p. Toutefois, les traitements locaux adaptatifs, qui tendent à se généraliser au sein des dispositifs mobiles, requièrent des ressources de calcul nettement plus importantes. En effet, un démosaïquage par la méthode d'*Hamilton & Adams*, un filtrage médian, bilatéral ou une correction gamma locale nécessitent souvent plus de dix milliards d'opérations par seconde.

TAB. 3.1: RESSOURCES DE CALCUL REQUISES POUR CHAQUE ALGORITHME, EXPRIMÉES EN OPÉRATIONS PAR PIXEL ET EN GOPS POUR DES FLUX VIDÉO HD.

Algorithme	Ressources Op./pixel	HD 720p GOPs	HD 1080p GOPs
Normalisation d'histogramme	23	0,45	1,19
Estimation et correction de la balance des blancs	34	0,67	1,76
Débruitage par filtre médian 3×3	$130 - 678$	$2 - 13{,}18$	$6 - 35{,}26$
Débruitage par filtre médian (sur histrogramme) 3×3	35	0,68	1,81
Filtre bilatéral 5×5	232	13,61	36,29
Suppression des pixels défectueux 3×3	45	0,87	2,33
Convolution 3×3	48	0,93	2,48
Convolution 5×5	137	2,68	7,15
Correction gamma (LUT)	23	0,45	1,19
Correction gamma locale adaptative 3×3	150	2,91	7,78
Correction gamma locale adaptative 5×5	403	7,84	20,89
Correction gamma locale adaptative 3×3 (LUT)	59	1,14	3,06
Correction gamma locale adaptative 5×5 (LUT)	148	2,86	7,67
Démosaïquage bilinéaire	95	1,84	4,92
Démosaïquage par constance de teinte	314	6,10	16,28
Démosaïquage par *Hamilton & Adams*	412	8,01	21,35
Changement d'espace de couleur	31	0,60	1,61

Le même décompte que précédemment est réalisé en fonction des différents « rôles » définis, et est présenté en Table 3.2. Cette méthode permet de différencier la part des ressources effectivement dédiée au calcul, à l'accès aux données et au contrôle. Celles affectées au calcul sont visibles en Table 3.3. Il apparaît que la part des ressources de calcul effectivement dédiée à l'obtention du résultat du traitement d'image représente moins de la moitié des ressources globalement nécessaires à l'exécution de l'algorithme alors que les fonctions d'accès aux données sont les plus coûteuses. Rappelons que la plupart des architectures décrites dans le chapitre précédent s'efforcent de décharger les unités de calcul de l'accès aux données, notamment grâce au DMA, ou encore avec la mise en place de hiérarchie mémoire ou de gestionnaires adaptés.

Ressources pour la chaîne complète de traitement Afin de déterminer les ressources de calcul nécessaires à l'exécution complète des chaînes d'amélioration d'image, nous avons appliqué l'étude algorithmique aux trois chaînes de références définies dans le premier chapitre. La Figure 3.3 confronte les ressources globales à celles effectivement dédiées au calcul nécessaire à l'exécution de chacun des traitements qui composent ces trois chaînes algorithmiques.

La première chaîne de référence montre que sur les 17,59 GOPs nécessaires à son exécution, seuls 6,53 GOPs sont dédiés au calcul. La seconde chaîne algorithmique de référence nécessite 70,89 GOPs pour traiter des images HD 1080p. Seuls 23,34 GOPs sont utilisés pour le calcul du résultat du traitement d'image, la plus grande partie restante étant affectée à l'accès aux données (calcul d'adresse et chargement des valeurs des pixels). De la même manière, la troisième chaîne utilise 7,25 GOPs sur un total de 18,86 GOPs pour exécuter les opérations correspondant au calcul du résultat. Dans les trois cas, la plus grande part des ressources est utilisée pour gérer les accès aux données, 30 % à 40 % pour le calcul du résultat du filtre et moins de 10 % pour le contrôle qui concerne essentiellement le parcours de l'image, la détermination de la couleur du pixel etc.

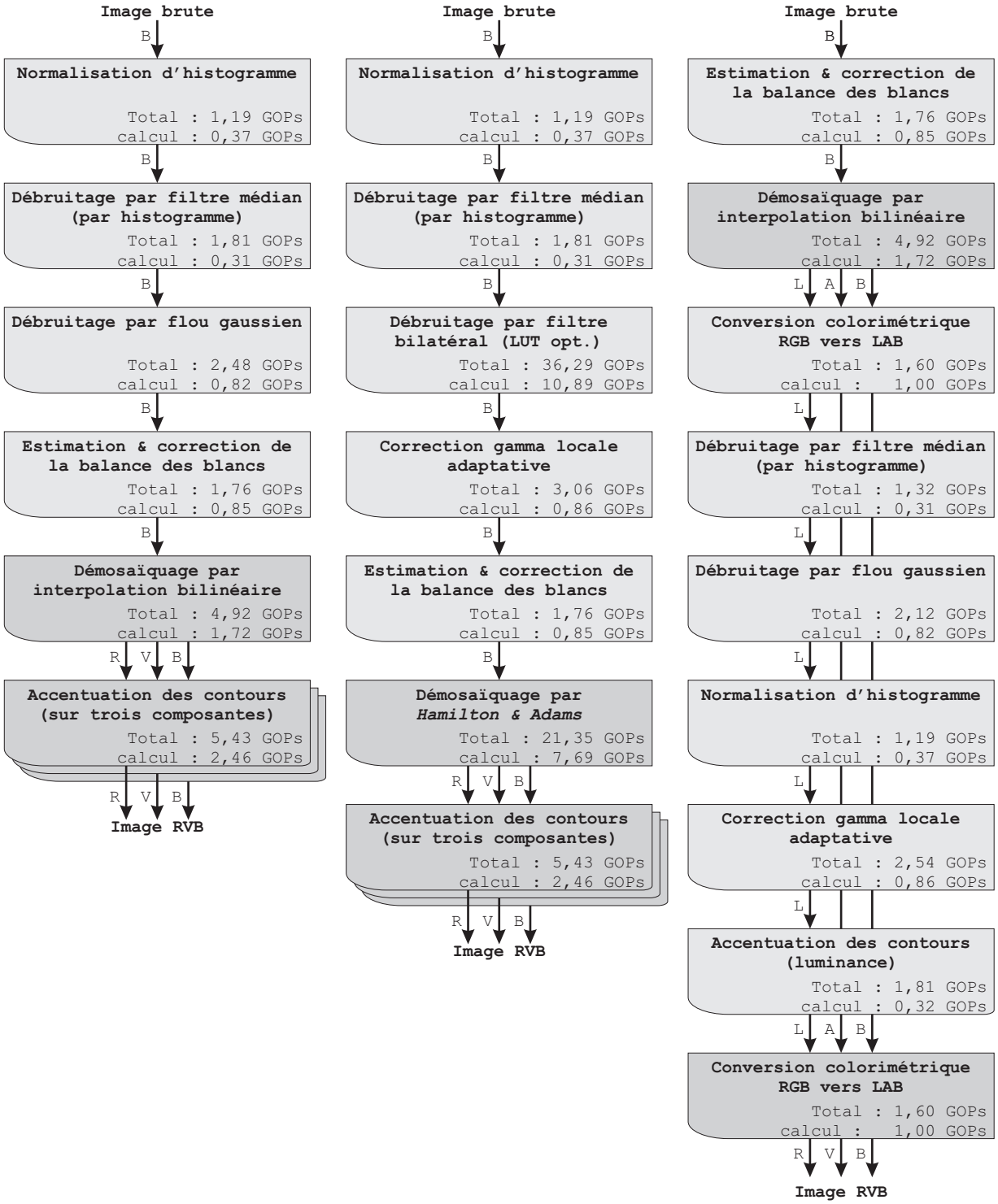


FIG. 3.3: CAPACITÉS DE CALCUL NÉCESSAIRES À L'EXÉCUTION DES CHAÎNES D'AMÉLIORATION D'IMAGE DE RÉFÉRENCE INTRODUITES DANS LE PREMIER CHAPITRE.

TAB. 3.2: RÉPARTITION DES RESSOURCES DE CALCUL EN FONCTION DES RÔLES ASSIGNÉS AUX OPÉRATIONS.

Algorithme	Calcul	Accès aux données	Contrôle
Normalisation d'histogramme	31 %	59%	10%
Estimation et correction de la balance des blancs	48 %	26%	17%
Débruitage par filre médian (tri)	39 %	37%	24%
Débruitage par filre médian (histrogramme)	17 %	64%	18%
Filtre bilatéral 5×5	30 %	51%	11%
Suppression des pixels défectueux	65 %	18%	12%
Convolution 3×3	33 %	53%	14%
Convolution 5×5	35 %	52%	14%
Correction gamma (LUT)	31 %	60%	9%
Correction gamma locale adaptative 3×3	61 %	27%	12%
Correction gamma locale adaptative 5×5	62 %	26%	12%
Correction gamma locale adaptative 3×3 (LUT)	28 %	57%	15%
Correction gamma locale adaptative 5×5 (LUT)	27 %	58%	15%
Démosaïquage bilinéaire	35 %	45%	10%
Démosaïquage par constance de teinte	42 %	47%	11%
Démosaïquage par Hamilton & Adams	36 %	41%	22%
Changement d'espace de couleur	62 %	29%	9%

Conclusion sur les ressources de calcul requises Cette étude montre que parmi les dizaines de GOPs nécessaires à l'exécution des algorithmes de la chaîne de reconstruction d'images, la part consacrée au contrôle est de loin la plus faible. Il est également remarquable que seulement le tiers des opérations est dédié au calcul effectif du résultat alors que plus de la moitié d'entre-elles est utilisé pour l'accès aux données. Il est donc essentiel de déterminer les types de données à manipuler et leurs modes d'accès.

3.1.2.2 Détermination des données manipulées et de leurs dynamiques

Afin de déterminer si les ressources de calcul dédiées à l'accès aux données sont similaires d'un algorithme à l'autre, nous nous proposons de déterminer le format de leurs entrées et de leurs sorties tout au long de la chaîne d'amélioration d'image. Les algorithmes sélectionnés sont par nature adaptés au mode flot de données. Or, le traitement d'images dans ce mode peut être vu comme le parcours séquentiel pixel par pixel des images. Cette étude nous conduit à recenser les types d'image utilisés en sortie des capteurs jusqu'à la reconstruction des images.

Les images brutes Les systèmes de capture couleurs mono-capteur largement utilisés dans le domaine de l'embarqué utilisent un filtre de couleur placé devant la surface photosensible. Ce filtre permet la capture des différentes couleurs sur la matrice photosensible du capteur. L'agencement des couleurs de ce filtre le plus connu est le filtre de Bayer qui a été présenté au premier chapitre, mais d'autres filtres existent et sont de plus en plus utilisés par les fabricants. Ce filtre permet de mesurer les différents niveaux d'éclairement pour chaque couleur primaire. L'image directement issue du capteur est une image brute (ou image CFA), qui ne possède pas les trois plans couleurs par pixel.

La plupart des traitements usuels peuvent être adaptés à moindre coût pour être réalisés sur une image brute. Pour cela, il est nécessaire de séparer les composantes de l'image, par exemple en trois plans couleur. Le calcul des adresses permettant d'accéder aux valeurs des pixels doit donc tenir compte du format du filtre de couleur

TAB. 3.3: RESSOURCES DE CALCUL REQUISES POUR LA PARTIE CALCUL DE CHAQUE ALGORITHME, EXPRIMÉES EN OPÉRATIONS PAR PIXEL ET EN GOPS POUR DES FLUX VIDÉO HD.

Algorithme	Ressources Op./pixel	HD 720p GOPs	HD 1080p GOPs
Normalisation d'histogramme	7,13	0,14	0,37
Estimation et correction de la balance des blancs	16,32	0,32	0,85
Débruitage par filtre médian (tri)	264,42	5,14	13,71
Débruitage par filtre médian (histrogramme)	5,95	0,12	0,31
Filtre bilatéral 5×5	69,60	4,08	10,89
Suppression des pixels défectueux	29,25	0,57	1,52
Convolution 3×3	15,84	0,31	0,82
Convolution 5×5	47,95	0,93	2,49
Correction gamma (LUT)	7,13	0,14	0,37
Correction gamma locale adaptative 3×3	91,50	1,78	4,74
Correction gamma locale adaptative 5×5	249,86	4,86	12,95
Correction gamma locale adaptative 3×3 (LUT)	16,52	0,32	0,86
Correction gamma locale adaptative 5×5 (LUT)	39,96	0,78	2,07
Démosaïquage bilinéaire	33,25	0,65	1,72
Démosaïquage par constance de teinte	131,88	2,56	6,84
Démosaïquage par *Hamilton & Adams*	148,32	2,88	7,69
Changement d'espace de couleur	19,22	0,37	1,00

utilisé. Ceci se traduit par l'écriture d'un segment de code différent pour chaque position dans ce filtre. Pour obtenir une image couleur à pleine résolution, il est nécessaire d'interpoler les valeurs des pixels « manquants ».

Les images couleurs Les images couleur sont obtenues après le démosaïquage des images brutes issues des systèmes mono-capteurs. Les images couleur peuvent aussi directement être issues de systèmes tri-capteurs (un capteur est utilisé pour chaque composante) ou des imageurs de type *Foveon*. Quel que soit leur espace colorimétrique, les images couleur comprennent généralement trois composantes. Le plus souvent, un même traitement est réalisé sur les différentes composantes séparément. Certains algorithmes de la chaîne d'amélioration d'image peuvent ne traiter que la composante de luminance afin de limiter les ressources de calcul, tandis que d'autres peuvent nécessiter l'accès à plusieurs composantes pour calculer la valeur d'un pixel. C'est par exemple le cas pour le changement d'espace colorimétrique ou pour certaines techniques de rehaussement des contours.

D'autres techniques sont employées pour représenter et traiter les images. Il s'agit par exemple du traitement d'image dans le domaine fréquentiel ou dans le domaine compressé que nous ne traiterons pas ici. Dans le cas du domaine fréquentiel, un nombre complexe comprenant partie entière et partie imaginaire constitue les échantillons. Par conséquent, il est nécessaire de pouvoir gérer des données d'au moins trois composantes, chacune étant associée à un plan de couleurs. Cela permet aussi d'introduire la possibilité d'associer des résultats intermédiaires aux flux de données. La Table 3.4 résume les différents formats d'entrées et de sorties possibles pour chaque algorithme analysé.

Les pixels et leurs voisinages L'analyse algorithmique a montré que les filtres nécessitent la valeur du pixel à traiter, mais bien souvent celles des pixels de son voisinage. En effet, nombreux sont les algorithmes basés sur l'utilisation d'une fenêtre glissante parcourant séquentiellement l'ensemble de l'image. Les tailles

TAB. 3.4: TAILLE DES MASQUES DE VOISINAGES TYPIQUEMENT UTILISÉS, ET FORMAT DES IMAGES D'ENTRÉE ET DE SORTIE DES ALGORITHMES ÉTUDIÉS. `B` POUR IMAGE BRUTE, `C1` POUR UN SEUL PLAN COULEUR OU EN NIVEAUX DE GRIS ET `C3` POUR LES IMAGES COMPORTANT TROIS PLANS (EX *RVB*, *YUV* ETC.)

Algorithme	Format d'entrée	Format de sortie	Voisinages
Normalisation d'histogramme	B, C1, C3	B, C1, C3	1×1
Estimation et correction de la balance des blancs	B, C3	B, C3	1×1 à 5×5
Débruitage par filtre médian (tri)	B, C1, C3	B, C1, C3	2×2 à 11×11
Débruitage par filtre médian (histogramme)	B, C1, C3	B, C1, C3	2×2 à 11×11
Filtre bilatéral 5×5	B, C1, C3	B, C1, C3	2×2 à 11×11
Suppression des pixels défectueux	B, C1, C3	B, C1, C3	2×2 à 11×11
Convolution 3×3	B, C1, C3	B, C1, C3	3×3
Convolution 5×5	C1, C3	C1, C3	5×5
Correction gamma (LUT)	C1, C3	C1, C3	3×3
Correction gamma locale adaptative 3×3	C1, C3	C1, C3	3×3
Correction gamma locale adaptative 5×5	C1, C3	C1, C3	5×5
Correction gamma locale adaptative 3×3 (LUT)	C1, C3	C1, C3	3×3
Correction gamma locale adaptative 5×5 (LUT)	C1, C3	C1, C3	5×5
Démosaïquage bilinéaire	B	C3	5×5
Démosaïquage par constance de teinte	B	C3	5×5
Démosaïquage par *Hamilton & Adams*	B	C3	5×5
Changement d'espace de couleur	C3	C3	1×1

typiques des voisinages pour le traitement de résolutions allant jusqu'à 1920×1080 recensées par notre étude sont présentées en Table 3.4. Cette liste n'est pas exhaustive et la plupart des algorithmes présentés ici sont couramment implémentés avec des tailles de voisinage plus importantes, notamment lorsque la résolution de l'image augmente.

Dynamique des données La dynamique des données correspond à celles des signaux d'entrées et de sorties associés aux traitements. Les valeurs des pixels sont couramment encodées sur des nombres entiers non signés de 8 et 12 bits de dynamique. Actuellement, les capteurs bas coût du marché utilisent des convertisseurs analogique-numérique 10 bits et tendent à atteindre 12 bits. L'image capturée ne couvre généralement pas l'ensemble de la dynamique suite à des erreurs de mesure d'exposition. Une correction est donc souvent nécessaire. De plus, elle est généralement réduite à 8 bits par composante pour répondre aux normes usuelles de compression d'image. Concernant les interfaces de sorties, la norme High Definition Multimedia Interface (HDMI) actuelle permet de supporter des échantillons de 8, 12 et 16 bits, bien que ces derniers soient réservés aux dispositifs professionnels de haute qualité d'image et des applications de pre-processing. Nous chercherons donc à traiter des échantillons de 8, 12 et 16 bits.

Dynamique des calculs Comme la surface silicium est une contrainte majeure, et que les échantillons sont codés sur des nombres entiers non signés, nous avons choisi d'implémenter les algorithmes étudiés à l'aide de nombres entiers signés. Différentes méthodes permettent d'adapter des algorithmes décrits en utilisant des nombres flottants vers des algorithmes utilisant des nombres entiers ou à virgule fixe [Belanovic 2005, Hauser 2001]. Il s'agit de méthodes analytiques ou par simulation [Menard uary, Coors 2002, Aamodt 2008, Menard 2008], et leur étude exhaustive dépasse le cadre de ces travaux. Elles sont cruciale pour le dimension-

nement correct d'une architecture embarquée où l'utilisation d'opérateurs manipulant des nombres à virgule flottante est proscrite en raison de leur encombrement silicium.

Afin d'offrir une précision suffisante dans les calculs portés, et ainsi ne pas induire une dégradation perceptible des images, nous avons déterminé par la simulation la dynamique minimale des opérateurs à intégrer. Pour cela, la procédure décrite dans la première partie de ce chapitre a été appliquée. Elle consiste à faire varier itérativement la taille des opérateurs en étudiant les dépassements de capacité et la dégradation des images par rapport à l'algorithme de référence. Nous avons ainsi pu déterminer que les opérateurs devaient présenter une dynamique de 16 bits pour supporter la plupart des traitements simples sur des pixels de 8 bits. Cette taille passe à 22 bits lorsque les données à traiter sont encodées sur 12 bits.

Conclusion sur les accès aux données Alors que nous avons pu voir que la chaîne d'amélioration d'image était naturellement adaptée aux traitements en mode flux, les traitements des images brutes nécessitent une adaptation des algorithmes à l'arrangement du filtre de couleur utilisé. Nous avons vu que de nombreux algorithmes nécessitent l'accès au voisinage du pixel à traiter, comme c'est traditionnellement le cas avec l'utilisation de fenêtres glissantes. La tailles de ces fenêtres de voisinages restent modestes, généralement à 5×5, bien qu'il soit parfois nécessaire de les augmenter pour que certains algorithmes, notamment de débruitage ou de corrections locales, restent efficaces. Le choix de la taille des fenêtres repose donc sur les programmeurs. C'est pourquoi cette caractéristique de l'architecture doit être flexible.

Les données issues des capteurs sont généralement encodées sur 12 bits, puis réduites à 8 bits par composante couleur. La taille des opérateurs doit être supérieure à 22 bits pour assurer une précision suffisante des calculs, même si une dynamique de 16 bits permet l'exécution d'une grande partie des algorithmes. C'est d'ailleurs le choix de la plupart des architectures vues dans le chapitre précédent. Toutefois, les valeurs des pixels présentent trois composantes, ainsi nous choisirons de préconiser l'utilisation de mots de 24 bits, ce qui permet d'intégrer trois valeurs de 8 bits – rouge, vert et bleu – tout en assurant les 22 bits de dynamique requis pour les calculs. Il est donc préférable de maintenir cette caractéristique paramétrable dans la mesure du possible.

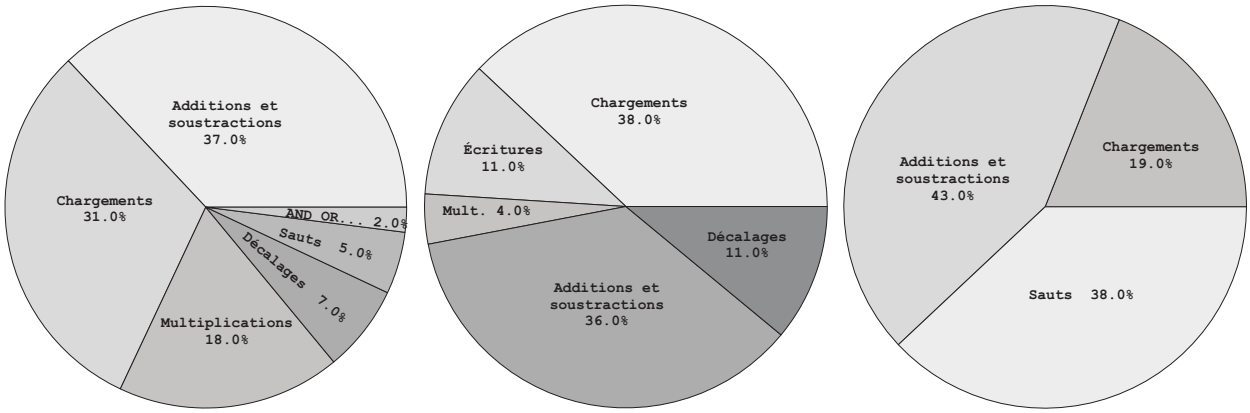
3.1.2.3 Recensement des opérations utilisées

Afin de déterminer le jeu d'opérateurs indispensable à l'exécution des algorithmes de traitement d'image, nous calculons le taux d'utilisation de chaque opération élémentaire lors de l'exécution des programmes. Pour cela, nous nous appuyons sur les résultats du profilage dynamique réalisé à l'aide des outils MAsS présentés précédemment en 3.1.1. En effet, ce profil permet de déterminer pour chaque exécution les opérations appelées selon leur « rôle » (calcul, accès aux données, contrôle). Les résultats présentés correspondent à la moyenne des résultats obtenus sur les trois chaînes de référence.

Répartition des opérations par « rôle » La Figure 3.4 représente la répartition des opérateurs utilisés pour chacun des rôles. Il apparaît sur la Figure 3.4a que 36 % des opérations nécessaires au calcul sont des additions et des soustractions, et 18 % sont des multiplications. Les copies de registre à registre représentent 31 % des opérations exécutées. En effet, nous avons considéré que les opérations devaient être systématiquement réalisées de registre à registre. La Figure 3.4b montre que les opérations d'accès aux données sont essentiellement constituées d'opérations simples puisque 30 % sont des copies de valeurs, 36 % des additions et soustractions, et 11 % des opérations de décalage. Elles sont principalement dédiées aux calculs d'adresses pour lesquels quelques multiplications interviennent. Les opérations de contrôle représentées en Figure 3.4c, consistent en la gestion des boucles de parcours de l'image ou de la fenêtre glissante, mais aussi de tests et de sauts permettant

de démarrer les segments de codes ad-hoc. On peut voir que 43 % sont des additions et soustractions, utilisées pour les compteurs et les comparaisons. Ces dernières induisent 38 % de sauts et seulement 19 % de copie de registre à registre.

Conclusion sur les opérations utilisées La partie calcul nécessite une plus grande diversité d'opérateurs que les autres « rôles », elle requiert aussi une flexibilité maximale. Or, nous avons vu que les algorithmes de traitement d'image peuvent être implémentés à l'aide d'opérateurs arithmétiques et logiques usuels. Une architecture de calcul se basant sur ce type d'opérateurs présente une flexibilité qui permet d'exécuter tous les types de traitements. Des opérations composites ont pu être identifiées, comme la valeur absolue, la recherche de *min* ou de *max* etc.. Bien que l'utilisation de ces opérations composites accélère notablement certains algorithmes, comme le filtre médian par exemple, leur utilisation moyenne sur l'ensemble d'une chaîne algorithmique n'est que de quelques pourcents. L'accès aux données est comparable d'un algorithme à l'autre, les variations sont généralement dues aux différents types d'images à traiter et à la taille des masques de voisinages manipulés. Les traitements de la partie contrôle apparaissent comme particulièrement réguliers et semblables d'un algorithme à l'autre.



(a) Calcul des résultats (b) Accès aux données (c) Opérations de contrôle

FIG. 3.4: RÉPARTITION DES DIFFÉRENTES OPÉRATIONS EN FONCTION DE LEURS « RÔLES » RESPECTIFS.

3.1.2.4 Étude des niveaux de parallélisme

Nous avons vu précédemment que la capacité de calcul nécessaire pour l'exécution des algorithmes qui constituent la chaîne de reconstruction et d'amélioration d'image peut dépasser plusieurs dizaines de milliards d'opérations par seconde pour le traitement d'un flux vidéo HD 1080p, ce qui équivaut à plusieurs centaines d'opérations par pixel. Cette capacité de calcul est directement liée au nombre d'échantillons à traiter par unité de temps. Ainsi, un flux vidéo, dont les trames sont de résolution $Largeur_{image} \times Hauteur_{image}$ et qui est cadencé à « C » images par seconde correspond à un flux de $Pixel_{Clk}$ échantillons qui doivent être traités par seconde. $Pixel_{Clk}$ est obtenu par l'équation 3.1, sans la prise en compte des délais d'intertrame et de fin de ligne.

$$Pixel_{Clk} = Largeur_{image} \times (Hauteur_{image} \times C) \tag{3.1}$$

Prenons par exemple une structure de calcul fonctionnant à 233 MHz et pouvant réaliser une opération par cycle. Si elle doit traiter un flux HD 1080p cadencé à 25 images par seconde ($Pixel_{Clk} = 51{,}8\ MHz$), alors seules

4 opérations par pixel peuvent être exécutées comme le montre l'équation 3.2. Or nous avons vu en 3.1.2.1 que plusieurs dizaines à plusieurs centaines d'opérations par pixel sont nécessaires au traitement vidéo HD. Par conséquent, il est crucial d'exploiter au mieux ces quelques cycles disponibles afin de réaliser un maximum d'opérations tout en maintenant une structure de calcul programmable, basse consommation et à faible surface silicium. C'est pourquoi, nous recherchons les différentes formes de parallélisme qu'il est possible d'exploiter d'un point de vue architectural.

$$NbOPpPx = \frac{Proc_{Clk}}{Pixel_{Clk}} \tag{3.2}$$

3.1.2.5 Détermination du parallélisme spatial

La majorité des traitements étudiés sont non récursifs, autrement dit, le résultat du traitement réalisé sur le premier pixel n'a pas d'influence sur le résultat du traitement réalisé sur le pixel voisin, et ainsi de suite. Ainsi les graphes d'exécution obtenus par notre analyse algorithmique sont semblables à celui qui est présenté en Figure 3.5. On peut voir sur celui-ci que la même suite d'instructions peut être réalisée simultanément sur les pixels successifs.

Il s'agit là des propriétés d'une structure de calcul SIMD pour laquelle chaque pixel est associé à une unité de calcul. Le parallélisme spatial inhérent au traitement d'image peut donc être exploité de cette façon. De la même manière, il peut être possible de regrouper le traitement de plusieurs pixels sur une même unité de calcul. Ces regroupements peuvent être réalisés par zone, région d'intérêt etc.

Avec une telle structure, p pixels peuvent être traités simultanément par p unités de calcul. Pour disposer de plus de temps pour réaliser le traitement (à horloge processeur $Proc_{Clk}$ égale), il est nécessaire de réduire la

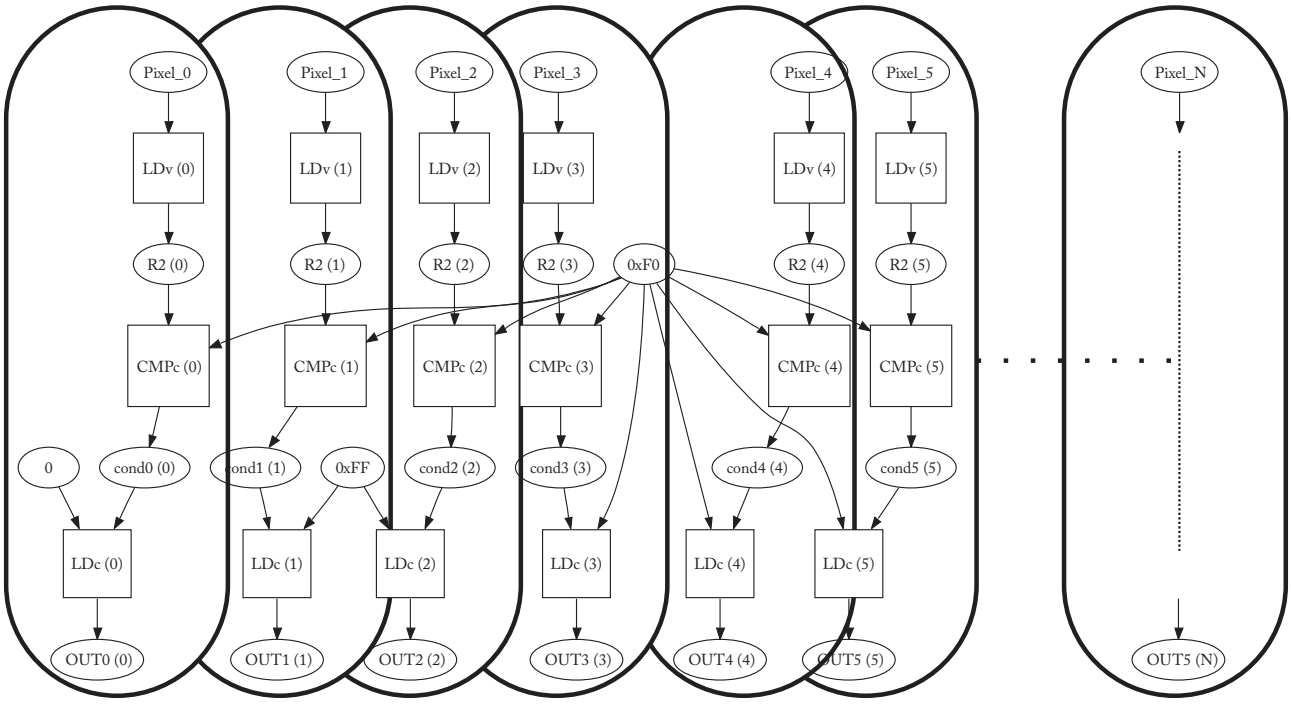


FIG. 3.5: MISE EN ÉVIDENCE DU PARALLÉLISME SPATIAL SUR LE GRAPHE D'EXÉCUTION D'UN SEUILLAGE, ON VOIT QU'IL EST POSSIBLE D'EXÉCUTER LES TRAITEMENTS SUR CHAQUE PIXEL SUCCESSIF INDÉPENDAMMENT LES UNS DES AUTRES.

cadence du flux, ce qui peut être fait en le parallélisant sur les p unités de calcul, comme on le voit sur l'équation 3.3 donnant le nombre d'opérations maximum pouvant être réalisées par pixel (si les unités de calcul exécutent une seule opération par cycle).

$$NbOPpPx_{SIMD=p} = \frac{Proc_{Clk}}{\frac{Pixel_{Clk}}{p}} \tag{3.3}$$

3.1.2.6 Détermination du parallélisme de tâches

Calcul et accès aux données L'étude algorithmique que nous avons réalisée montre que près de la moitié des opérations sont exécutées pour réaliser un accès aux données. De plus, la plupart des traitements consistent à accéder à la valeur d'un pixel ou aux valeurs des pixels de son voisinage. Cette fonctionnalité peut donc être séparée de la partie effectuant le calcul du résultat qui représente entre 30 % et 40 % du temps de traitement mais qui nécessite un accès à ces données.

Il s'agit donc de mettre à disposition pour chaque unité de calcul la valeur du pixel à traiter, mais aussi celles des pixels de son voisinage lorsque cela est nécessaire. Ceci peut être réalisé par un module dédié comme c'est le cas pour la plupart des architectures décrites dans l'état de l'art. Le maximum d'efficacité est obtenu si cette opération est réalisée parallèlement au calcul, de manière transparente et en une durée négligeable devant le temps de calcul. Ceci peut par exemple être réalisé en préparant les données pour le calcul du pixel suivant pendant que les calculs sur le pixel courant sont en cours d'exécution. Dans ce cas, seul le temps effectivement dédié aux opérations de calcul du résultat impacte les performances du système. Toutefois, le choix et le chargement des valeurs à traiter parmi celles du voisinage rendu disponible dépendent de l'algorithme à exécuter. Par conséquent, cette partie de l'accès aux données doit être maintenue programmable. Les résultats des calculs intermédiaires n'étant que rarement exploités pour le calcul des autres pixels, l'accès au voisinage peut être en lecture seule dans l'optique de réduire la surface silicium, la consommation électrique ou les temps d'accès.

Opérations de contrôle L'étude précédente a montré que près de 10% des opérations sont dédiées au contrôle du programme, c'est à dire à la gestion du parcours de l'image, au démarrage des segments de code correspondant aux traitements à réaliser etc. La majeure partie de ces opérations sont similaires d'un traitement à l'autre. Elles peuvent donc être réalisées en temps masqué par des modules adaptés.

Enchaînement des traitements Les traitements de chaîne d'amélioration d'image peuvent naturellement être enchaînés les uns aux autres de telle manière que lorsque le premier pixel est traité, il puisse aussitôt faire l'objet du traitement suivant, et libérer les ressources pour que le pixel suivant prenne immédiatement sa place. Il est donc crucial d'exploiter au mieux cette propriété afin de mettre en œuvre une architecture réalisant en parallèle les traitements qui composent la chaîne algorithmique.

3.1.2.7 Détermination du parallélisme au niveau instructions

Afin de maximiser le nombre d'opérations à réaliser par cycle d'horloge, une structure SIMD de p unités de calcul programmables est proposée – il s'agit donc de p processeurs. Pour ces derniers, il peut s'avérer efficace d'exploiter le parallélisme au niveau instructions en leur intégrant plusieurs voies de calcul. C'est pourquoi, il est nécessaire d'évaluer au préalable avec précision à quel point il peut être exploitable. Pour cela, nous cherchons à déterminer le taux de remplissage des voies de calcul en fonction de leur nombre. En effet, il est inutile d'utiliser de la surface silicium pour implémenter des voies de calcul qui ne seraient que rarement utilisées.

Le nombre d'opérations par pixel qu'il est possible d'exécuter au meilleur cas, en fonction du nombre de processeurs p SIMD et de voies qu'il intègre est donné par l'équation 3.4. Nous nous appuyons d'abord sur la littérature pour déterminer si cela est envisageable, puis sur une étude approfondie des algorithmes représentatifs de la chaîne de traitement d'image.

$$NbOPpPx_{SIMD=p;VLIW=Nbvoies} = \frac{Proc_{Clk}}{\frac{Pixel_{Clk}}{p \times Nbvoies}} \tag{3.4}$$

Le parallélisme au niveau instructions dans la littérature Les premiers travaux faisant référence datent des années 70, où il est montré que le parallélisme d'instructions moyen est proche de 2 [Tjaden 1970]. Par la suite, des travaux de *Digital DEC* montrent qu'il est raisonnable de compter sur un parallélisme au niveau instructions compris entre 2 et 4 [Wall 1991, Jouppi 1989] en fonction des algorithmes utilisés. Les études réalisées convergent généralement vers un parallélisme au niveau instructions compris entre 1,5 et 3, mais qui peut atteindre 6 en fonction des bancs de tests et des types de données manipulées [Hung 1999, Rau 1993]. Par exemple *Fisher* a obtenu un parallélisme au niveau instruction de 4 [Fisher 1984] lors de la conception du premier processeur VLIW. Ce type de parallélisme est fortement dépendant de la nature des données manipulées, des algorithmes, du mode de compilation, mais aussi de certains éléments architecturaux comme les mécanismes de prédiction de branchement.

Concernant les études récentes, nous retenons celle réalisée par *Salamì San Juan* [Juan 2007], qui aborde, dans ses travaux de thèse, quel peut être le nombre de voies optimal pour un processeur VLIW traitant du multimédia. Ses résultats montrent que le passage de deux voies à quatre voies permet d'obtenir une accélération pouvant atteindre 1,6×, alors que le passage à huit voies ne permet plus d'obtenir un gain très significatif. Nous voyons aussi, dans cette même étude, que la taille de la file de registres est multipliée par plus de trois lorsque le nombre de voies passe à quatre. Ces résultats confirment d'autres travaux [Lopez 1998] qui prédisent une augmentation importante de la surface silicium et de la consommation électrique lorsque des files de registres sont conçues pour permettre plusieurs accès concurrents. Il est montré par *Rixner* [Rixner 2000] que la surface et la consommation des files de registres varient en N^3, N étant le nombre de voies d'accès en lecture.

Dans notre cas, les données manipulées sont des valeurs scalaires – nombres entiers signés. De plus, nos choix architecturaux s'orientent vers une structure SIMD avec les éléments de contrôle et d'accès aux données séparées des processeurs de calcul. Le programmeur ne décrivant que les cœurs de boucles de ses algorithmes, il est possible d'en déduire que l'utilisation de branchements sera marginale ou inexistante. Nous retenons donc entre 2 et 4 voies de calcul au sein de chaque processeur de calcul. Dans ce cas, ces processeurs de calcul seraient des processeurs VLIW. Afin de déterminer ce nombre de voies avec précision, une étude algorithmique est donc réalisée, d'abord en se basant sur une analyse statique des algorithmes, puis sur une analyse du graphe d'exécution issu de notre profilage dynamique précédemment décrit.

Détermination du parallélisme au niveau instructions par analyse statique Le laboratoire LCE a développé une méthode permettant de déterminer le parallélisme moyen d'un programme. Cette méthode se base sur une analyse statique du code assembleur obtenu après compilation pour un processeur de type Scalable Processor Architecture (SPARC). Comme nous avons choisi de séparer les tâches de calcul de celles d'accès aux données et de contrôle, le protocole de test a dû être adapté afin d'en tenir compte. Cette analyse permet de déterminer le taux de remplissage moyen de voies supposées homogènes en fonction de leur nombre, comme le présente la Figure 3.6. Les algorithmes ont été classés en trois catégories :

1. traitements réguliers : il s'agit des algorithmes basés sur la convolution, ou ayant des motifs réguliers;

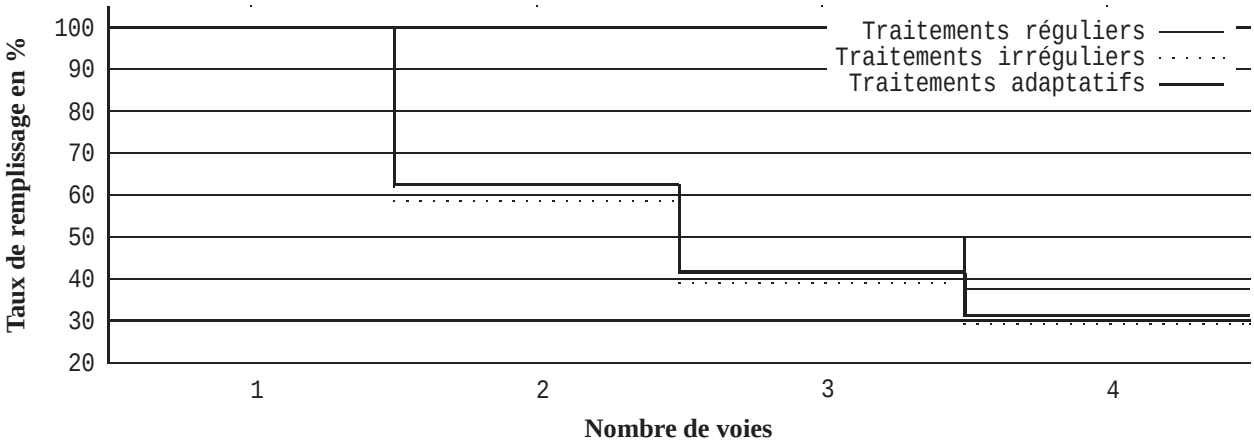


FIG. 3.6: TAUX DE REMPLISSAGE MOYEN DES VOIES DE CALCUL EN FONCTION DE LEUR NOMBRE – MÉTHODE PAR ANALYSE STATIQUE DES CODES APPLICATIFS, OPÉRATEURS DE REGISTRE À REGISTRE.

2. traitement irréguliers : il s'agit des algorithmes faisant appel à de nombreuses instructions conditionnelles (sauts) comme c'est le cas du filtre médian;
3. traitement adaptatifs : il s'agit des algorithmes locaux adaptatifs, du filtre bilatéral au démosaïquage de Hamilton Adams;

La Figure 3.6 montre que le taux de remplissage moyen de deux voies homogènes dépasse 60 %, tandis qu'il reste inférieur à 50 % si une troisième voie est ajoutée. Nous avons considéré que les opérations sont exécutées de registre à registre.

La même étude montre que le niveau de parallélisme diminue d'environ 15 % lorsque les opérateurs peuvent traiter directement des données sans devoir les stocker en file de registres au préalable. Ainsi, moins de voies de calcul sont nécessaires pour exécuter efficacement le programme, puisque seules deux voies peuvent être exploitées à plus de 50 % en moyenne.

Cette réduction est à corréler au fait que 31 % des opérations réalisées consistent à copier des valeurs de registre à registre. Ainsi, si on considère que les opérateurs ne sont pas contraints d'utiliser des opérandes issus de la file de registres, le nombre d'instructions du programme diminue, ce qui impacte le temps d'exécution et la consommation électrique. Dans ce cas, le processeur est orthogonal. Cela a pour conséquence de permettre de réduire le nombre de registres et par là même, la surface silicium et la consommation électrique. Le surcoût architectural induit reste à évaluer et à prendre en compte. Cette première étude nous permet donc de déterminer que deux voies de calcul sont suffisantes pour accélérer significativement les opérations de calcul en exploitant le parallélisme au niveau instructions. Afin de confirmer ces résultats qui souffrent de l'imprécision due à l'analyse statique, nous avons réalisé une analyse dynamique basée sur l'exploration du graphe d'exécution obtenu précédemment par les outils MAsS vus au début de ce chapitre.

Détermination du parallélisme au niveau instructions par analyse dynamique Pour confirmer les résultats obtenus précédemment, nous avons mis au point et implémenté au sein de la suite MAsS une méthode d'analyse du graphe d'exécution construit lors de l'analyse algorithmique. Cette méthode consiste en un parcours récursif des graphes d'exécution des traitements à analyser. La méthode est décrite dans l'algorithme de

la Figure 3.7 et consiste à rechercher, décompter puis supprimer les nœuds sources du graphe d'exécution, ce qui produit un nouveau graphe. Ces trois opérations sont répétées jusqu'à ce qu'il ne comporte plus de source – donc de nœud puisqu'il s'agit d'un graphe orienté. Le parallélisme moyen au niveau instructions correspond à la moyenne de chacun des décomptes consécutifs. La Figure 3.8 illustre cette méthode étape par étape en s'appuyant sur un exemple simple de convolution 2×2.

```
1: NbInstructions=0;
2: while graphe.existe do
3:   i=i+1 // incrément du compteur d'itérations;
4:   for Source = each(graphe.sources()) do
5:     NbInstructions=NbInstructions+1
6:     graphe.delete(Source)
7:   end for
8: end while
9: ILP=NbInstructions/i;
10: return ILP;
```

FIG. 3.7: ALGORITHME D'ESTIMATION DU NIVEAU DU PARALLÉLISME MOYEN AU NIVEAU INSTRUCTION POUR UN GRAPHE D'EXÉCUTION DONNÉ.

La version utilisée de l'algorithme intègre les mécanismes permettant d'attribuer des ressources en fonction des opérateurs matériels que l'on souhaite rendre disponibles. Pour cela, il suffit de définir un jeu d'opérateurs à utiliser, lorsque un nœud source est identité et qu'il correspond à un opérateur disponible de ce groupe, il est supprimé, décompté, et l'opérateur correspondant est marqué comme utilisé jusqu'à l'itération suivante. Afin de déterminer le nombre de registres optimal à utiliser, cette technique est étendue aux registres qui font partie du graphe d'exécution. Il est ainsi possible de répéter l'analyse pour différents arrangements d'opérateurs ainsi que pour différentes tailles de la file de registres.

L'analyse est d'abord réalisée en considérant que les algorithmes sont décrits en utilisant des opérateurs requérant des opérandes stockés en file de registres. La Figure 3.9 représente les résultats de cette analyse en montrant le taux d'utilisation des voies en fonction de leur nombre. On voit dans ce cas que 3 voies de calcul peuvent être utilisées avec un taux de remplissage moyen compris entre 40 % et 50 %.

Cette analyse montre que 84 % des opérandes utilisés sont stockés en file de registres et issus de résultats d'opérations précédentes (le chargement d'une donnée externe en file de registre est considéré comme une opération). C'est pourquoi, l'analyse est répétée avec des algorithmes décrits en considérant que le jeu d'instructions est orthogonal. Les résultats obtenus font l'objet de la Figure 3.10. Dans ce cas, le nombre d'instructions est réduit de plus de 23 % en moyenne, la Figure 3.11 illustre cette réduction du nombre d'instructions en confrontant les deux modes de programmation. Lorsque les algorithmes sont ré-écrits de cette manière, les données externes ne sont plus recopiées en file de registres avant d'être l'objet de calculs, ainsi le nombre d'accès aux données externes augmente, et seules 63 % des données sont déjà en file de registres lorsque les opérateurs en ont besoin. Seule une opération sur 14 (7,1 %) en moyenne nécessite l'accès à deux opérandes externes.

L'utilisation de trois voies de calcul par processeur ne permet pas d'augmenter significativement (7 %) les performances globales lorsque le jeu d'instructions est orthogonal. Leur taux de remplissage moyen est compris

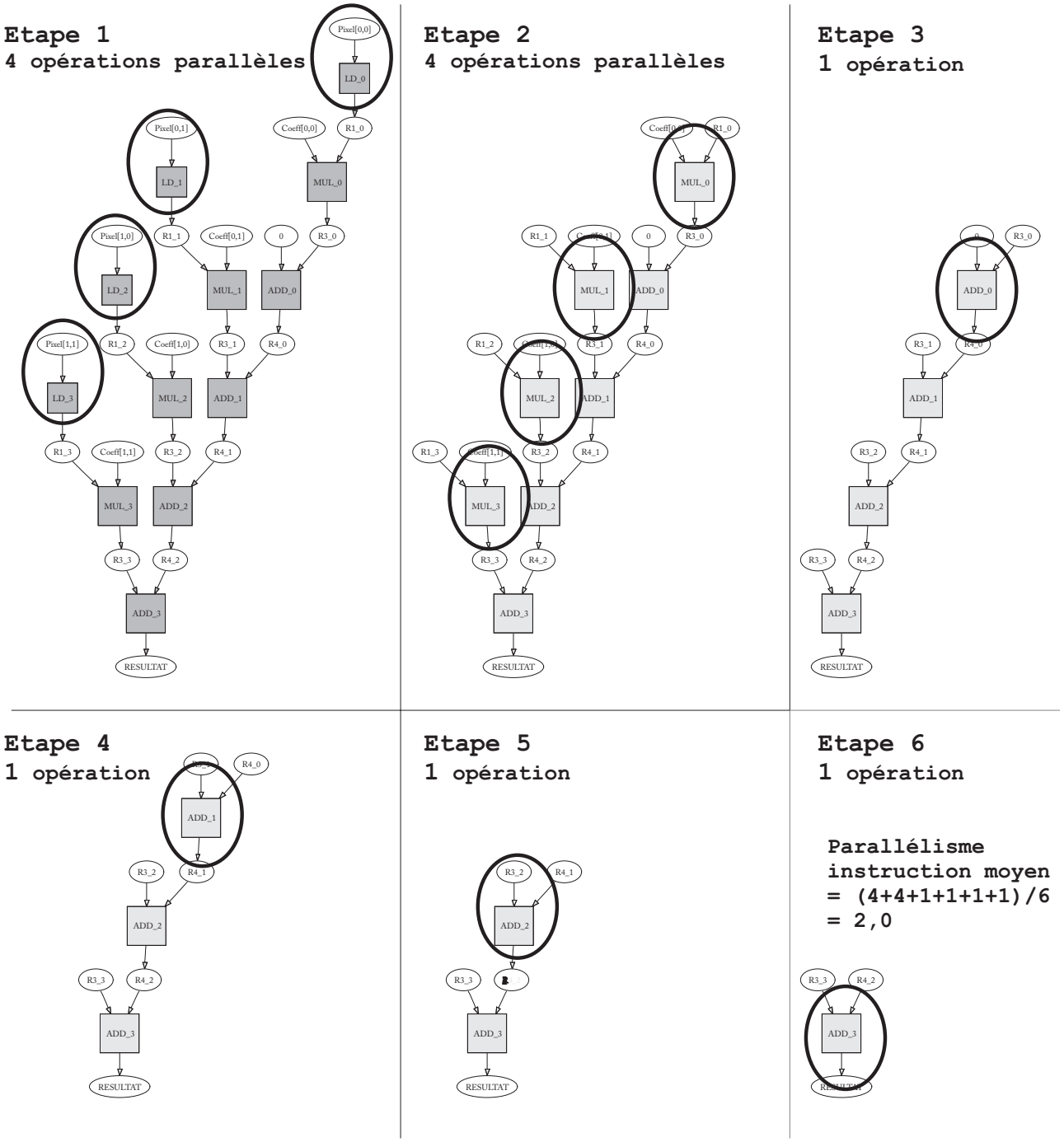


FIG. 3.8: MÉTHODE EMPLOYÉE POUR L'EXTRACTION DU PARALLÉLISME AU NIVEAU INSTRUCTIONS.

entre 26 % et 35 %, alors qu'il est compris entre 58 % et 68 % pour deux voies.

Le nombre de registres effectivement sollicités est compris entre 3 et 11 en fonction des algorithmes utilisés. Nous choisirons donc une file de registres réduite entre huit et seize registres pour chaque processeur élémentaire. Enfin, il apparaît que l'utilisation de deux additionneurs/soustracteurs permet d'augmenter significativement le taux d'utilisation moyen des deux voies (18 %), tandis que la duplication d'autres opérateurs n'apparaît pas nécessaire pour augmenter de manière importante le taux d'utilisation des voies (moins de 9%

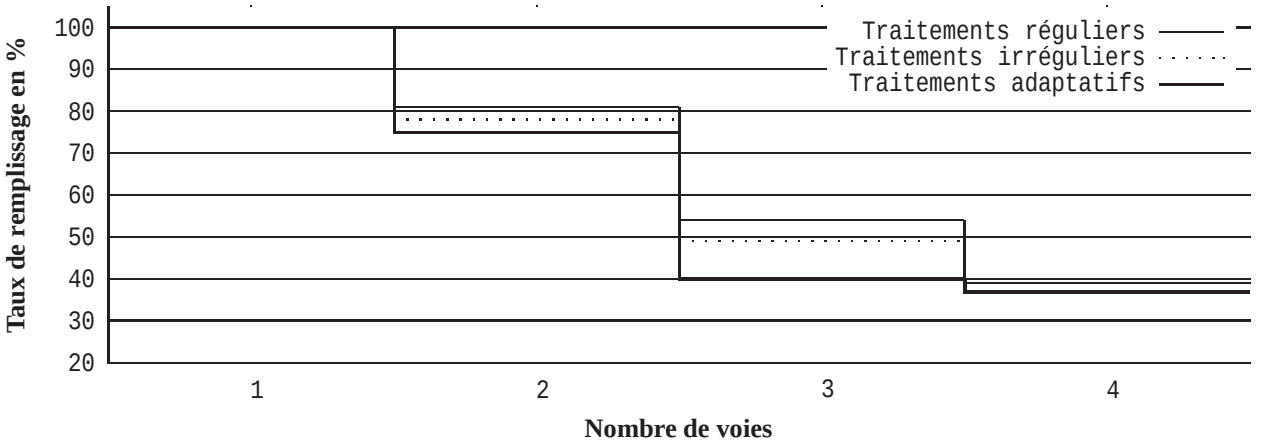


FIG. 3.9: TAUX DE REMPLISSAGE MOYEN DES VOIES DE CALCUL EN FONCTION DE LEUR NOMBRE – MÉTHODE PAR ANALYSE DES GRAPHES D'EXÉCUTION DES APPLICATIONS, OPÉRATEURS DE REGISTRE À REGISTRE.

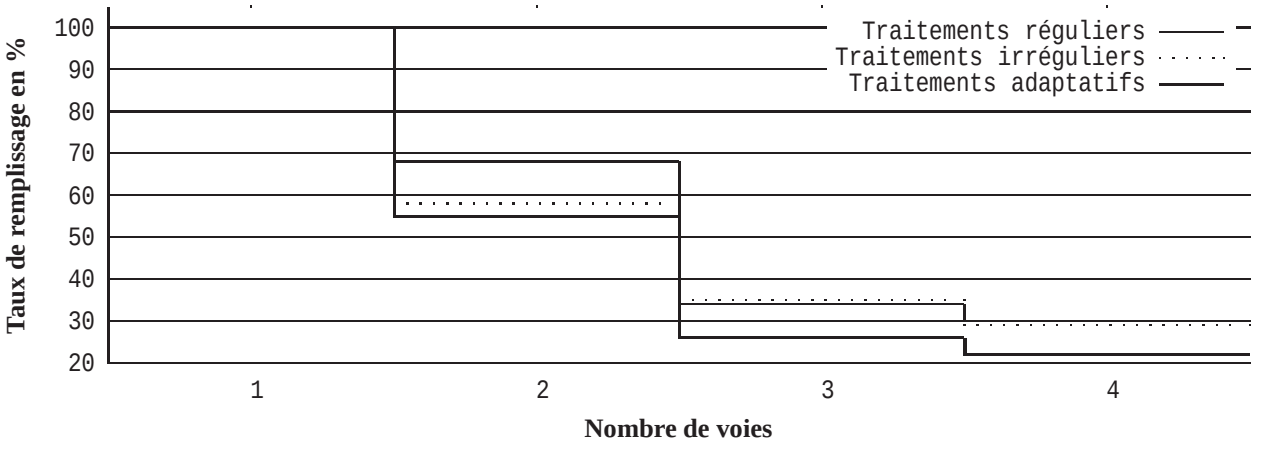


FIG. 3.10: TAUX DE REMPLISSAGE MOYEN DES VOIES DE CALCUL EN FONCTION DE LEUR NOMBRE – MÉTHODE PAS ANALYSE DES GRAPHES D'EXÉCUTION DES APPLICATIONS, OPÉRATEURS À ACCÈS DIRECT.

lorsque le multiplieur et les affectations sont dupliquées).

$Y = A \times x + B$

```
R1 ← A                          R1 ← A × x
R2 ← x
R1 ← R1 × R2
R2 ← B                          R1 ← R1 + b
R1 ← R1 + R2
```

FIG. 3.11: A GAUCHE : PROGRAMME UTILISANT DES OPÉRATEURS REGISTRE À REGISTRE, À DROITE, LE MÊME PROGRAMME UTILISANT DES OPÉRATEURS À ACCÈS DIRECT.

Conclusion sur l'analyse du parallélisme au niveau instructions L'étude réalisée permet de déterminer le nombre de voies à utiliser au sein des processeurs. Ces résultats montrent que l'utilisation de trois voies composées d'opérateurs traitant de registre à registre est envisageable, tandis que l'utilisation de deux voies est efficace lorsque le jeu d'instructions est orthogonal. Ainsi, plusieurs choix résumés en Table 3.5 sont possibles. À ce stade de notre démarche, la surface et la consommation ne peuvent pas être calculées avec précision dans la mesure où trop peu d'éléments architecturaux sont fixés. Ces informations sont estimées qualitativement.

TAB. 3.5: RÉSUMÉ DES CHOIX ARCHITECTURAUX POSSIBLES POUR LE PROCESSEUR ÉLÉMENTAIRE.

Choix	Nombre de voies	Accès aux opérandes	Vitesse normalisée	Taux d'utilisation	Surface et consommation	Bilan
1	1	R→R	1,00	100 %	1	++
2	1	*→R	1,23	100 %	1	+++
3	2	R→R	1,38	78 %	4	++++
4	2	*→R	1,47	64 %	5	+++++
5	3	R→R	1,48	45 %	9	++
6	3	*→R	1,56	31 %	10	+

Nous optons pour la solution (N°4 de la Table 3.5) qui consiste en un processeur élémentaire VLIW deux voies, pour lequel les opérations peuvent être réalisées depuis n'importe quelle source de données – sans contraintes d'opérations de registre à registre Afin de réduire sa surface silicium, il est envisagé de simplifier ce choix. Les solutions utilisant plus de trois voies de calcul sont exclues pour le gain en vitesse qui ne justifie pas, à nos yeux, la complexité architecturale qui en découle.

Conclusion sur l'analyse algorithmique

Les ressources nécessaires à l'exécution des chaînes d'amélioration d'image sont de plusieurs dizaines de milliards d'opérations par seconde. Les cadences devant être soutenues sont de plusieurs dizaines de millions de pixels par seconde. Le second chapitre nous a permis d'évaluer différentes architectures de calcul, aussi bien dédiées qu'entièrement programmables. Nous avons vu que les architectures programmables conventionnelles ne permettent pas de produire de telles capacités de calcul, alors que les architectures parallèles de type SIMD et particulièrement celles en mode flux sont adaptées au calcul à de telles cadences.

L'analyse algorithmique a montré que les traitements peuvent être enchaînés les uns à la suite des autres. De plus, la séparation des fonctions d'accès aux données, de contrôle et de calcul proprement dit permet d'exploiter ce niveau supplémentaire de parallélisme. Les fonctions d'accès aux données sont similaires d'un algorithme à l'autre et elles représentent près de la moitié des opérations requises à l'exécution d'un programme. Il est

donc envisagé de les réaliser sous forme de modules dédiés puisque les traitements accèdent systématiquement aux valeurs du voisinage autour du pixel sur lequel opérer. La taille de ce voisinage est généralement comprise entre 1×1 et 11×11 en fonction de l'algorithme utilisé. La valeur des pixels est généralement codée sur 10-12 bits en sortie du capteur, et tend à passer à 8 bits au fur et à mesure de l'exécution des algorithmes, aussi une dynamique de 16 à 24 bits permet d'assurer des calculs de traitement d'image avec une précision acceptable. Comme pour les fonctions d'accès aux données, les fonctions de contrôle peuvent être séparées de la partie calcul.

Les modules dédiés au calcul doivent conserver un haut niveau de flexibilité en restant entièrement programmables. Pour cela, nous avons choisi d'utiliser des processeurs VLIW comportant deux voies de calcul, ce qui nous permet d'exploiter le parallélisme au niveau instructions. Nous avons montré que l'utilisation de deux voies était un compromis entre vitesse d'exécution et complexité architecturale. Il est maintenant nécessaire de proposer un modèle architectural correspondant à ces caractéristiques.

3.2 Proposition d'un modèle architectural

La section précédente nous a permis de déterminer les différentes propriétés propres aux classes d'algorithmes que l'architecture doit supporter, certaines de ces propriétés donnent lieu à des orientations architecturales. Cette section se propose de traduire ces orientations en un modèle architectural capable de répondre aux contraintes imposées par notre cahier des charges, que ce soit en termes de surface silicium, de consommation électrique ou en termes de flexibilité et de capacité de calcul disponible.

Les choix réalisés concernant le processeur élémentaire destiné à exécuter la partie calculatoire des algorithmes sont détaillés dans la première partie de cette section. Une architecture de calcul parallèle s'appuyant sur ces processeurs est ensuite présentée, suivie du mode d'accès aux données capable de supporter un flux vidéo HD en temps réel et de le distribuer aux différents processeurs. La quatrième partie de ce chapitre introduit comment les processeurs de calcul sont contrôlés tandis que la dernière partie expose comment ces éléments peuvent être interconnectés entre eux pour permettre un maximum de flexibilité tout en limitant la surface silicium. La quatrième partie de ce chapitre introduit comment les processeurs de calcul sont contrôlés tandis que la dernière partie expose comment ces éléments peuvent être interconnectés entre eux pour permettre un maximum de flexibilité tout en limitant la surface silicium.

3.2.1 Proposition d'un modèle de processeur de calcul

Les processeurs de calcul exécutent la partie calculatoire des traitements. Comme ils sont destinés à fonctionner en mode SIMD, il est préférable de limiter leur surface silicium tout en permettant l'exécution d'un maximum d'opérations par cycle d'horloge.

3.2.1.1 Nombre de voies et file de registres

Nous avons vu que l'utilisation d'un modèle VLIW deux voies est un compromis permettant un gain notable sur le temps d'exécution par rapport à l'utilisation d'un processeur scalaire classique. Ces deux voies peuvent être homogènes, ce qui revient à dupliquer tous les opérateurs, et permet d'exploiter au mieux le parallélisme d'instructions. Utiliser des voies hétérogènes présente une plus grande efficacité silicium, mais rend complexe l'exploitation du parallélisme d'instructions.

Nous proposons d'opter pour une solution consistant à intégrer un jeu d'opérateurs et à le mutualiser pour que les deux voies puissent les utiliser. Le nom de *SplitWay* qui lui a été attribué est tiré de cette caractéristique éponyme. L'accès aux opérateurs est symétrique, même si les deux voies ne peuvent réaliser simultanément la même opération, à moins d'instancier deux fois le même opérateur. Cette solution peut introduire un surcoût en surface lié à l'interconnexion des opérateurs, c'est pourquoi des opérateurs propres à chacune des voies peuvent être implémentés. De la même manière, nous choisissons de mutualiser la file de registres pour les deux voies, ce qui permet un maximum de flexibilité pour le programmeur. De même, des registres propres à chacune des voies peuvent être ajoutés.

3.2.1.2 Sources de données

Nous avons vu précédemment que l'utilisation d'opérateurs de registre à registre nécessite l'exécution d'un grand nombre d'opérations d'affectation de données vers la file de registres. C'est pourquoi nous privilégions des opérateurs capables d'utiliser des opérandes issus indifféremment de la file de registres comme de n'importe quelle autre source, ce qui signifie que le processeur est orthogonal. Les types de sources de données que l'ensemble des opérateurs doivent être en mesure d'exploiter sont les suivantes :

- valeurs de la file de registres;
- valeurs de pixels;
- valeurs constantes;
- valeurs de LUTs;
- valeurs stockées en mémoire;
- valeurs issues de coprocesseurs;
- valeurs issues de processeurs voisins;
- données internes à l'architecture (registres spéciaux etc.);
- données externes à l'architecture.

Pour que les différents opérandes issus des différentes sources puissent être sélectionnés et transmises aux opérateurs, il est nécessaire de dupliquer la logique d'accès pour chacun d'eux, ce qui peut s'avérer coûteux en surface silicium. Or, l'étude réalisée précédemment a montré que 85 % des instructions utilisent un des deux opérandes dont la valeur est déjà stockée en file de registres. C'est pourquoi nous proposons d'implémenter la structure d'accès aux données externes seulement sur le premier opérande de chacune des voies. Par ailleurs, les résultats des calculs sont directement stockés dans des registres prédéfinis de la file de registres, ainsi lorsque le traitement arrive à son terme, ces valeurs sont automatiquement sauvegardées.

3.2.1.3 Les opérateurs nécessaires

Afin de rendre possible le portage des différentes classes d'algorithmes de traitement étudiées, il est souhaitable d'être en mesure d'exécuter les opérations suivantes :

1. addition et soustraction;
2. affectation;
3. multiplications;
4. décalages;
5. opérations booléennes;
6. accès à une LUTs.

Notons qu'aucune de ces opérations n'est spécifique au traitement d'image, ce qui rend le processeur de calcul utilisable pour tout type de traitement. Comme il peut être envisagé d'étendre le jeu d'instructions ultérieurement, nous avons choisi de rendre le modèle de processeur totalement paramétrable. De nombreux traitements

utilisent des résultats pré-calculés dans des LUTs pour accélérer leur exécution, c'est pourquoi il est nécessaire d'intégrer cette fonctionnalité, par exemple, par l'utilisation d'une mémoire interne au processeur.

L'étude algorithmique a montré que l'utilisation d'opérateurs d'une dynamique de 24 bits permet l'exécution des traitements recensés. L'utilisation d'opérateurs 16 bits, bien que suffisante pour la plupart des cas, ne permet pas de couvrir tous les algorithmes. C'est pourquoi nous préconiserons une dynamique de 24 bits pour les opérateurs du processeur, qu'une implémentation suffisamment générique permettra de réduire ou d'augmenter avant la création du circuit.

3.2.1.4 LUTs et mémoire de travail

Pour permettre l'utilisation de LUTs ou d'une mémoire de travail, un espace mémoire doit être prévu. L'analyse algorithmique montre que les traitements peuvent utiliser jusqu'à trois LUTs pour précalculer des opérations. Dans de nombreux cas, la taille de ces LUTs peut être réduite en ne stockant qu'une partie des valeurs recalculées.

Une mémoire de travail est plus rarement nécessaire, généralement utilisée pour les traitements à base d'histogramme. De nombreux algorithmes étudiés ne nécessitent pas ou très peu d'espace mémoire (quelques mots) pour être exécutés. Toutefois, la présence d'un espace mémoire ou la possibilité d'en intégrer, augmente la capacité de l'architecture à recevoir des algorithmes complexes. C'est pourquoi nous prévoirons un espace d'adressage suffisamment conséquent pour ne pas limiter l'architecture dans ses évolutions futures.

Conclusion sur le modèle de processeur

Le processeur que nous proposons d'utiliser est un processeur VLIW deux voies. Afin de limiter la surface silicium, les opérateurs sont mutualisés entre les deux voies du chemin de données d'où son nom de processeur *SplitWay*. Nous préconisons l'utilisation d'opérateurs de 24 bits de dynamique afin d'assurer une précision suffisante pour l'ensemble des calculs. Un espace mémoire permet l'utilisation de LUTs ou d'une mémoire de travail.

3.2.2 Proposition de regroupement des processeurs SplitWay

L'étude algorithmique a montré que le parallélisme spatial inhérent aux images était exploitable en associant un pixel à chaque processeur de calcul fonctionnant en mode SIMD.

3.2.2.1 Mode de fonctionnement

Les mêmes instructions sont transmises simultanément à l'ensemble des processeurs fonctionnant en mode SIMD. L'étude algorithmique montre que la plupart des algorithmes qui traitent des images brutes exécutent un cœur de boucle différent en fonction de la couleur du pixel considéré. De même, certains algorithmes réalisent des traitements en fonction du résultat d'un calcul intermédiaire, comme par exemple l'algorithme de démosaïquage de *Hamilton & Adams*. Si une structure SIMD est utilisée, alors la totalité des cas possibles doit être transmise aux processeurs. Ceux-ci évaluent s'ils doivent ou non exécuter les instructions en vérifiant systématiquement que les conditions nécessaires sont remplies. Ces conditions peuvent par exemple être la position du pixel dans la matrice de couleur quand il s'agit de traitements sur des images brutes, ou bien un résultat précédent associé à la valeur du pixel à traiter.

```
Si (couleur=Rouge)                    |
     1: Opération 1 Rouge             |  1: Si Rouge Opération 1 Rouge Sinon RIEN
     2: Opération 2 Rouge             |  2: Si Rouge Opération 2 Rouge Sinon RIEN
     ...                              |  ...
     R: Opération N Rouge             |  R: Si Rouge Opération N Rouge Sinon RIEN
                                      |
Sinon (couleur=Vert)                  |
     1: Opération 1 Vert              |  1: Si Vert Opération 1 Vert   Sinon RIEN
     2: Opération 2 Vert              |  2: Si Vert Opération 2 Vert   Sinon RIEN
     ...                              |  ...
     V: Opération V Vert              |  V: Si Vert Opération V Vert   Sinon RIEN
                                      |
Sinon (couleur=Bleu)                  |
     1: Opération 1 Bleu              |  1: Si Bleu Opération 1 Bleu   Sinon RIEN
     2: Opération 2 Bleu              |  2: Si Bleu Opération 2 Bleu   Sinon RIEN
     ...                              |  ...
     B: Opération B Bleu              |  B: Si Bleu Opération B Bleu   Sinon RIEN
Fin SI                                |
                                      |
Total : Max (R,V,B) + tests           |  Total : R+V+B
```

FIG. 3.12: EXEMPLE DE CODE TRAITANT DES IMAGES BRUTES, DÉCRIT AVEC DES BLOCS CONDITIONNELS À GAUCHE, ET DES OPÉRATIONS CONDITIONNELLES DE TYPE SIMD À DROITE.

La Figure 3.12 confronte schématiquement l'écriture « standard » et l'écriture SIMD d'une suite d'opérations pour le traitement d'images brutes. Dans cet exemple, trois segments de code distincts sont exécutés en fonction de la couleur du pixel à traiter. La Figure 3.13 illustre comment ce type de programme est exécuté sur une structure de calcul SIMD. Dans ce cas, toutes les instructions correspondant aux cas potentiels sont transmises aux différents processeurs. Il s'agit d'instructions conditionnelles, ainsi les processeurs exécutent seulement celles qui correspondent à leur cas d'exécution et inhibent les instructions correspondant aux autres cas. La durée d'exécution du programme est donc égale à la somme des durées de chaque segment qui le compose, tandis que dans le cas d'une structure programmable « standard », la durée d'exécution est celle du segment le plus long. Autrement dit, dans le cas du traitement d'une image brute, le nombre de cycles nécessaires à l'exécution d'un algorithme sur une image brute est égal à la somme des cycles nécessaires au traitement de chaque cas du filtre de couleur. Comme chaque processeur ne traite qu'une couleur à la fois et qu'il inhibe les instructions correspondant aux autres couleurs, il passe la majorité de son temps à attendre que les autres processeurs exécutent les opérations qui les concernent.

Pour remédier à cette limitation du mode SIMD, nous proposons l'utilisation d'une structure Multi-SIMD. Dans ce mode, les différents cas d'exécution possibles sont séparés les uns des autres et forment des segments de codes distincts. Seules les instructions correspondant aux cas à exécuter sont lues et transmises aux processeurs qui les exécutent alors. La lecture des instructions est réalisée en fonction de la valeur que prend un attribut associé aux données à traiter (couleur, position, résultats de calculs intermédiaires etc.). Cet exemple est illustré par la Figure 3.14 où on peut voir que chaque segment de code est décrit séparément par le programmeur en fonction de la couleur du pixel à traiter dans l'image brute. Dans cet exemple, le temps d'exécution du traitement correspond au temps d'exécution le plus long de chaque segment. La durée d'exécution des traitements peut donc être déterminée dès l'écriture ou à la compilation du programme et elle est complètement déterministe.

3.2.2.2 Budget en surface silicium et consommation électrique

L'estimation des ressources requises pour l'exécution des algorithmes a montré qu'une chaîne d'amélioration d'image peut nécessiter plusieurs dizaines de GOPs pour les parties dédiées aux calculs. En supposant que les processeurs VLIW deux voies sont utilisés à une fréquence $Processeur_{Clk}$, l'équation 3.5 permet de

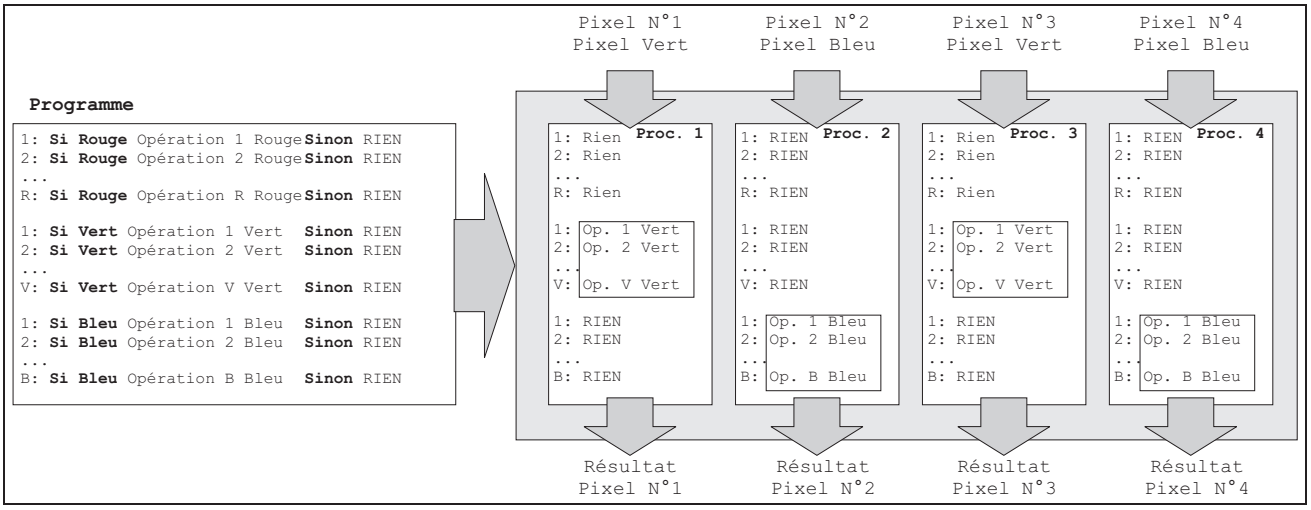


FIG. 3.13: EXEMPLE SCHÉMATIQUE DE L'EXÉCUTION D'OPÉRATIONS TRAITANT DES IMAGES BRUTES SUR UNE STRUCTURE SIMD.

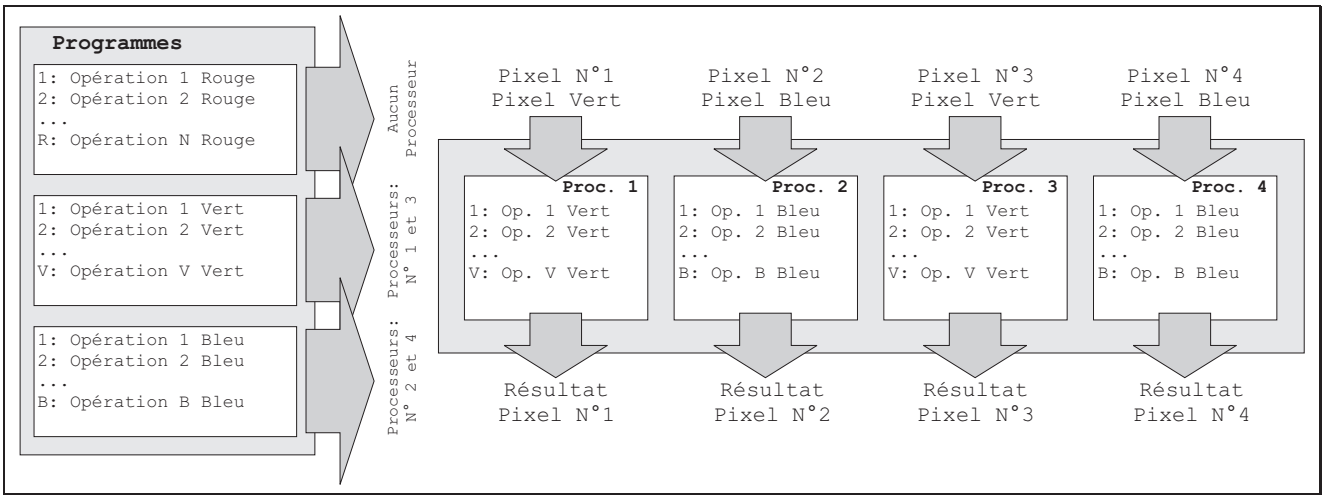


FIG. 3.14: EXEMPLE SCHÉMATIQUE DE L'EXÉCUTION D'OPÉRATIONS TRAITANT DES IMAGES BRUTES SUR UNE STRUCTURE MULTI-SIMD.

déterminer le nombre Nb_{Procs} de processeurs nécessaires à partir de la capacité de calcul à atteindre. Ainsi, il est nécessaire de disposer d'environ 50 processeurs *SplitWay* pour atteindre 20 GOPs de capacité de calcul disponible[1] à une fréquence fixée à 200 MHz[2] ce qui permet d'assurer le traitement HD 1080p à 25 images par seconde (52 MPixels/s).

$$Nb_{Procs} = p = \frac{RessourcesCalul}{2 \times Processeur_{Clk}} \tag{3.5}$$

En tenant compte des quelques millimètres carrés de surface silicium imposés par les contraintes du cahier des charges et des capacités d'intégration des fondeurs actuels, il est possible d'estimer le budget en portes

1. 20 GOPs correspond aux ressources de calcul nécessaires à l'exécution de la troisième chaîne de référence en considérant que les accès aux données et le contrôle sont séparés de la partie calcul.

2. L'état de l'art montre que les architectures offrant un meilleur compromis entre consommation électrique et capacité de calcul fonctionnent à une fréquence de l'ordre de 200 MHz.

logiques que le processeur *SplitWay* doit respecter. Pour cela, nous avons estimé le taux d'intégration en portes logiques par unité de surface de différents processeurs de l'état de l'art en technologie 65 nm. Notre estimation a été réalisée à partir des chiffres communiqués sur différents composants programmables basse consommation et montre qu'entre 200 à 500 kGates par mm ^{2} peuvent être envisagées, sachant que le fondeur *TSMC* annonce 854 kGates/mm^2 en 65 nm [TSMC 2007].

Si la surface silicium disponible est de 1,5 mm^2 pour l'intégration des processeurs, alors le budget est de l'ordre de 3 à 10 kGates, soit environ 0,03 mm^2. L'état de l'art montre qu'il est possible de parvenir à ce résultat, avec par exemple le processeur *APS-3* 32 bits [Cortus 2008] et ses 10 kGates pour une fréquence de fonctionnement annoncée à 500 MHz. Nous procédons de même pour la consommation électrique, en nous basant sur l'état de l'art pour définir un budget de 130 à 350 mW pour l'ensemble des processeurs de calcul, nous en déduisons que le budget en consommation électrique est de l'ordre de 4 à 7 mW par processeur de calcul.

3.2.2.3 Détermination des groupes de processeurs

Si l'ensemble des traitements est réalisé par le même groupe de processeurs, alors le parallélisme de tâches qui découle de leur enchaînement possible ne peut pas être exploité. C'est pourquoi, nous proposons de regrouper les processeurs de calcul en plusieurs entités que nous nommerons « tuiles de calcul ». Celles-ci sont destinées à exécuter un traitement donné sur un flux de pixels, puis générer un flux en sortie exploitable par une autre tuile de calcul comme le schématise la Figure 3.15.

Outre le fait d'exploiter le parallélisme de tâches en enchaînant les traitements, cette structure permet le partitionnement des algorithmes complexes sur plusieurs tuiles de calcul. De plus, chacune d'elles est supposée autonome, et peut donc être remplacée par une Propriété Intellectuelle (PI) dédiée. De même, des regroupements hétérogènes peuvent être envisagés pour lesquels des fréquences de fonctionnement différentes sont attribuées en fonction d'un compromis entre ressources de calcul nécessaires et consommation électrique, l'ensemble étant synchronisé par le flux de données.

L'étude algorithmique permet de voir qu'il est possible de regrouper plusieurs traitements ensemble (par exemple balance des blancs et démosaïquage). Pour assurer une flexibilité minimale, il est préférable de disposer d'au moins quatre à six tuiles de calcul. La majorité des traitements peut être exécutée avec quelques

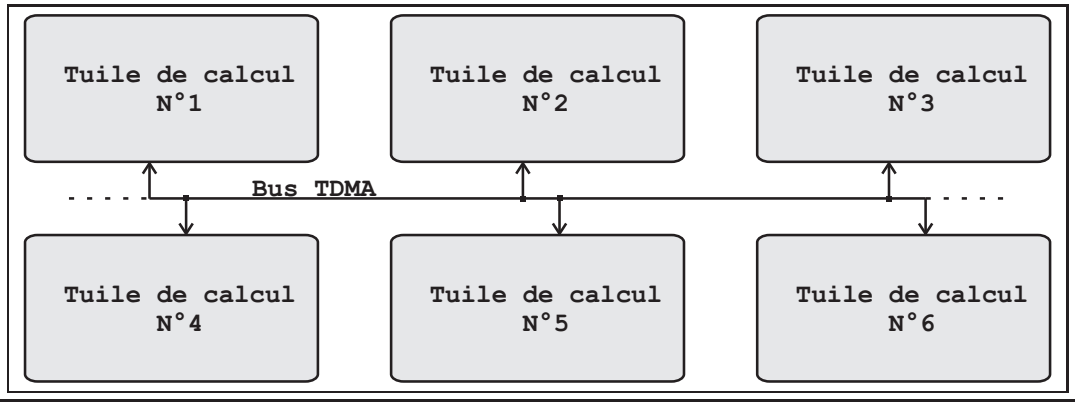


FIG. 3.15: EXEMPLE D'ENCHAÎNEMENT DE PLUSIEURS TRAITEMENTS SUR TUILES DE CALCUL REGROUPANT DES PROCESSEURS EN MODE MULTI-SIMD.

dizaines d'opérations par cycles. Il est réaliste de disposer d'une cinquantaine de processeurs de calcul pour leur exécution. Par exemple, en supposant qu'ils sont répartis uniformément en cinq tuiles de calcul de 10 processeurs chacune, leur capacité de calcul serait de 76 opérations par pixel (30 cycles par pixel) à 200 MHz, suffisante pour la plupart des traitements. Ceux d'entre-eux qui ne peuvent être supportés sur une seule tuile de calcul doivent être scindés ou simplifiés.

Deux méthodes peuvent être employées pour adapter la capacité de calcul d'une tuile. La première consiste, par reconfiguration, à intervenir sur le nombre de processeurs au sein d'une tuile de calcul. La seconde consiste à adapter la fréquence de fonctionnement des processeurs d'une même tuile avec pour conséquence d'adapter la capacité de calcul en fonction du traitement à exécuter.

Pour cela, la tuile de calcul peut être connectée à plusieurs horloges processeurs sélectionnées en fonction du compromis recherché entre performances et ressources de calcul. La durée d'exécution d'un programme étant connue et déterministe, il suffit de sélectionner la fréquence de fonctionnement la plus faible permettant d'assurer le traitement en temps réel. Il est possible, à partir de l'équation 3.6, de déterminer la fréquence de fonctionnement minimale à utiliser en fonction du nombre de processeurs Nb_{Procs}, de leur fréquence de fonctionnement $Proc_{Clk}$ et du nombre d'instructions (de cycles) par pixel que requiert l'algorithme pour son exécution.

$$Proc_{Clk} = \left\lceil \frac{NbInstr}{Nb_{Procs}} \right\rceil \times Px_{Clk} \tag{3.6}$$

Prenons par exemple, un flux vidéo HD 1080p cadencé à 52 MHz et traité par un programme nécessitant 14 cycles par pixel. Ainsi, pour une tuile de 10 processeurs cadencés à 200 MHz, 38 cycles par pixel sont disponibles pour l'exécution d'un traitement en temps réel, d'après l'équation 3.6. Le processeur passera donc 24 cycles à attendre la donnée suivante. Aussi, réduire sa fréquence de fonctionnement permet de l'utiliser pleinement tout en diminuant la consommation électrique de la tuile de calcul. Dans ce cas, une fréquence de fonctionnement de 80 MHz suffit à assurer le traitement en temps réel.

3.2.2.4 Communication entre processeurs

Certains algorithmes nécessitent d'accéder aux valeurs de résultats des calculs réalisés sur les pixels voisins. Ceux-ci sont connus par les processeurs voisins. Aussi est-il indispensable de permettre la communication entre processeurs voisins. Pour cela, différentes techniques peuvent être mises en œuvre. Par exemple un réseau d'interconnexion peut être mis en place, ou encore un bus. En considérant qu'en mode SIMD, toutes les demandes de communication sont réalisées simultanément et qu'elles doivent être satisfaites en un temps connu, nous préférons l'utilisation d'une connexion point à point entre processeurs. Une telle solution permet de limiter la surface silicium et la complexité architecturale qu'un bus ou qu'un réseau de communication engendre. Ainsi, chaque processeur peut lire les valeurs de ses voisins « est » et « ouest ».

3.2.2.5 Gestion des données composites

Définir des tuiles de calcul permet de leur spécifier des formats pour les données composites en entrée et en sortie. En effet, les données transmises dans le flux peuvent être composées de plusieurs valeurs de pixels (*RVB*, *LAB*) ou diverses données de travail que nous appellerons *métadonnées*. Celles-ci peuvent être les résultats de calculs intermédiaires utilisés pour le traitement suivant, ou encore des coordonnées (pour les applications d'interpolation *pan tilt* etc.) La décomposition de ces métadonnées peut être réalisée logiciellement par le programmeur, mais son impact est élevé en instructions supplémentaires (masques, décalages etc.).

Aussi proposons-nous que les formats des flux d'entrée et de sortie soient configurés au niveau de la tuile de calcul. Cela nécessite de rendre configurable les formats des masques de données. Lorsque le programmeur lit la valeur d'une donnée, qu'elle qu'en soit sa source, seule la partie du mot qu'il aura requise est transmise aux opérateurs. Ceci peut être réalisé de manière transparente comme le présente la Figure 3.16, où le 3ème champ est lu. L'exploitation du *Sub Word Parallelism* est en production depuis plusieurs années [Kittitornkun 2003, Lee 1996, Lee 1995], mais l'approche proposée diffère de celles habituellement implémentées puisqu'il ne s'agit pas de traiter simultanément différentes parties d'un mot.

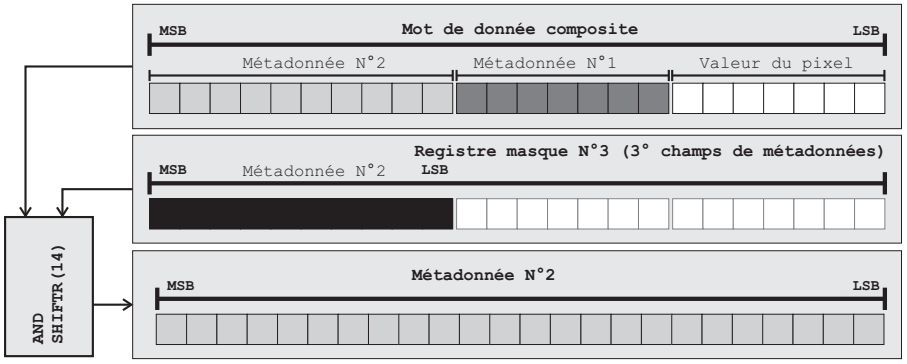


FIG. 3.16: EXEMPLE DE LECTURE D'UNE MÉTADONNÉE DE 10 BITS DANS UN MOT DE 24 BITS QUI CONTIENT 3 VALEURS. SA VALEUR EST AUTOMATIQUEMENT EXTRAITE À L'AIDE DU MASQUE DE LECTURE CONFIGURÉ AU PORTAGE DU TRAITEMENT

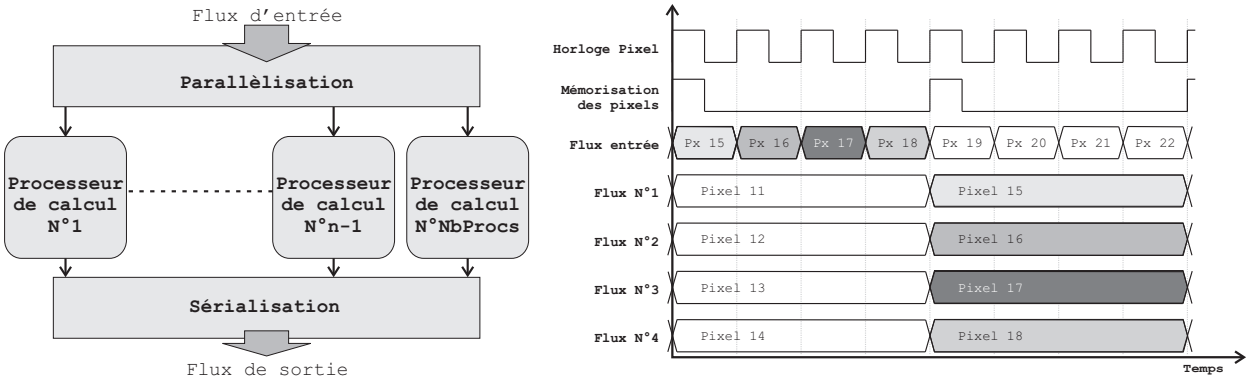
Comme pour la lecture, le programmeur précise à quelle position dans le mot il souhaite écrire ses données. De cette manière, aucun surcoût en nombre d'instructions n'est à déplorer pour la composition et la décomposition de données composites. Outre l'utilisation classique consistant à intégrer plusieurs composantes du pixel dans le même mot de données, le programmeur peut par exemple traiter et transmettre entre les tuiles de calcul des mots de données contenant des nombres complexes avec une partie réelle et une partie imaginaire, une partie entière et une partie décimale, ou encore des mots contenant la valeur du pixel associée à différents résultats de calculs intermédiaires.

Conclusion sur le regroupement des processeurs de calcul

Nous proposons de regrouper plusieurs processeurs *SplitWay* VLIW deux voies en plusieurs tuiles de calcul Multi-SIMD capables de traiter un flux de pixels. Chaque tuile est autonome et exécute le programme correspondant au(x) traitement(s) qui lui est(sont) associé(s). Elle peut être configurée pour traiter les formats de données composites souhaités par le programmeur. Ces données sont décomposées et recomposées de manière transparente pour le programmeur. Pour répondre aux contraintes de surface silicium et de consommation électrique, la taille de notre processeur de calcul doit être inférieure à 10 kGates pour une consommation électrique inférieure à 7 mW.

3.2.3 Définition du mode d'accès aux données

Nous avons vu que la séparation des fonctions d'accès aux données de la partie calcul permettait de réduire significativement les ressources nécessaires à leur exécution. Les fonctions d'accès aux données sont similaires pour l'ensemble des traitements étudiés, ce qui nous permet d'envisager l'utilisation de ressources dédiées. Il s'agit d'apporter directement aux processeurs de calcul les valeurs des pixels qu'ils doivent manipuler.



(a) PARALLÉLISATION D'UN FLUX DE PIXEL VERS Nb_{Procs} PROCESSEURS DE CALCUL.

(b) CHRONOGRAMME DE LA DISTRIBUTION D'UN FLUX DE PIXELS SUR $Nb_{Procs} = 4$ PROCESSEURS DE CALCUL.

FIG. 3.17: SCHÉMA ET CHRONOGRAMME DU PROCESSUS DE DISTRIBUTION DES PIXELS AUX DIFFÉRENTS PROCESSEURS D'UNE MÊME TUILE DE CALCUL. (a) SCHÉMA DE PRINCIPE DE LA PARALLÉLISATION DES PIXELS VERS LES Nb_{Procs} PROCESSEURS DE CALCUL (b) CHRONOGRAMME DES SIGNAUX CORRESPONDANT AUX FLUX PARALLÉLISÉS POUR $Nb_{Procs} = 4$ PROCESSEURS DE CALCUL, ON PEUT VOIR QUE LES PIXELS SONT DISPONIBLES PENDANT Nb_{Procs} PÉRIODES PIXELS.

3.2.3.1 Accès au pixel à traiter

La sous-section précédente a présenté l'organisation des processeurs au sein de tuiles de calcul enchaînées les unes à la suite des autres et fonctionnant en mode Multi-SIMD. Les Nb_{Procs} processeurs d'une tuile de calcul peuvent ainsi traiter simultanément Nb_{Procs} pixels, un pixel étant associé à chaque processeur. Pour cela, le flux de données est parallélisé, transmis aux processeurs puis sérialisé comme le présente la Figure 3.17a. Pour que les valeurs des pixels soient les mêmes durant toute la durée de l'exécution du traitement, elles sont mémorisées à la sortie du module de parallélisation tous les Nb_{Procs} pixels.

Comme le montre la Figure 3.17b, les valeurs des pixels à traiter sont disponibles pendant Nb_{Procs} cycles pixels, et automatiquement remises à jour tous les Nb_{Procs} pixels. C'est à ce moment que le signal signifiant le lancement du segment de programme correspondant au pixel à traiter est envoyé pour que les instructions soient lues en mémoire. Les images sont donc traitées par segment de Nb_{Procs} valeurs. Le nombre de cycles disponibles pour le traitement d'un pixel peut donc être obtenu par l'équation 3.7.

$$NbCycles = \left\lfloor \left(\frac{Processeur_{Clk}}{Pixel_{Clk}} \right) \times Nb_{Procs} \right\rfloor \tag{3.7}$$

3.2.3.2 Accès au voisinage d'un pixel

De nombreux traitements nécessitent généralement les valeurs des pixels avoisinant le pixel à traiter. Or, le mode de distribution présenté précédemment n'est adapté qu'à la distribution de pixels uniques sans leurs voisinages. De plus, la taille du voisinage à considérer dépend des algorithmes à traiter comme nous avons pu le voir précédemment en Table 3.4 de la page 52. Pour permettre cela, il est nécessaire de mémoriser les valeurs à transmettre dans des mémoires ligne avant de pouvoir reconstituer le voisinage à manipuler. Comme pour les pixels, ce dernier doit être accessible par les unités de calcul concernées durant toute la durée du traitement. C'est pourquoi nous proposons de remplacer le module de parallélisation vu précédemment par un « gestionnaire de voisinages » qui remplit le même rôle étendu au support de voisinages. Celui-ci doit être

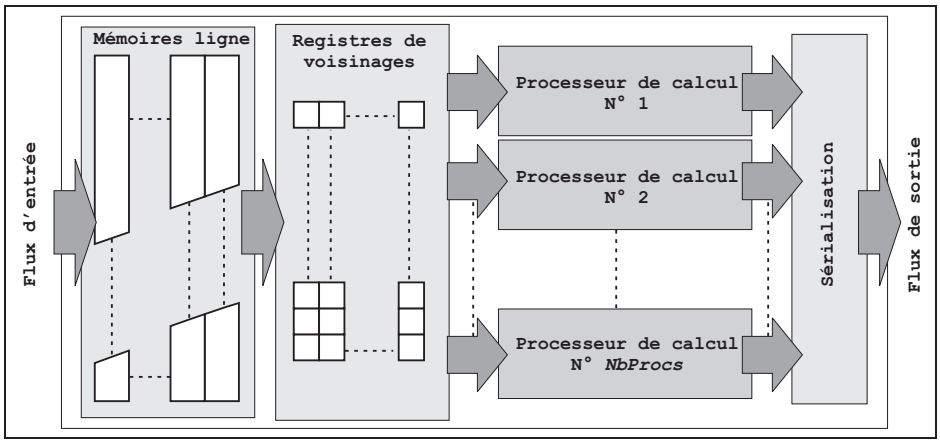


FIG. 3.18: HIÉRARCHIE MÉMOIRE PERMETTANT LA DISTRIBUTION DES VALEURS DES PIXELS ET DE LEURS VOISINAGES AUX PROCESSEURS D'UNE TUILE DE CALCUL.

adapté à la préparation et à la mise à disposition de voisinages de taille $N \times M$ aux processeurs de calcul, avec N le nombre de lignes et M le nombre de colonnes du masque de voisinage considéré. De plus, ce gestionnaire peut être configuré pour supporter différentes tailles de voisinages.

Comme ce voisinage est constitué de N lignes, il est nécessaire de mémoriser les valeurs des données du flux. Ainsi pour supporter un voisinage de taille 5×5, la mémorisation d'au moins 4 lignes et 5 pixels sera nécessaire avant de démarrer le traitement.

Il est toutefois impossible de mémoriser plus d'une trentaine de lignes HD 1080p de 1920 pixels de large sur une surface d'un millimètre carré. Comme un gestionnaire de voisinages est destiné à être intégré à chaque tuile de calcul – qui sont au moins au nombre de quatre –, la taille des voisinages qu'ils pourront supporter est naturellement limitée par la surface silicium qui leur est allouée. De plus, cette estimation de l'espace mémoire utilisable ne tient pas compte des composants permettant aux processeurs de calcul d'assurer l'accès aux valeurs des pixels de ces voisinages. En effet la surface qui leur est attribuée peut se révéler conséquente puisque Nb_{Procs} doivent accéder simultanément à ces données.

Comme Nb_{Procs} processeurs de calcul sont supposés accéder simultanément à des valeurs de pixels différentes stockées dans ces mémoires, elles doivent disposer de Nb_{Procs} ports de lecture. La surface silicium de tels bancs mémoire varie au carré du nombre de ports. C'est pourquoi, nous proposons d'utiliser des mémoires simple port pour la mémorisation des lignes. Celles-ci sont connectées à des « registres de voisinages » qui reconstituent l'ensemble d'un segment de pixels à traiter, comme l'illustre la Figure 3.18.

Les processeurs peuvent directement lire l'ensemble des valeurs des pixels qui restent inchangées pendant toute la durée du traitement. De la même manière, un composant dédié, associé ou remplaçant le processeur de calcul, peut accéder simultanément et instantanément à toutes les valeurs du voisinage à traiter. Pendant que les processeurs réalisent le traitement d'un segment de Nb_{Procs} valeurs, les valeurs des registres de voisinages peuvent être préparées en temps masqué.

Conclusion sur l'accès aux données

Afin de distribuer le flux de données sous la forme de voisinages directement exploitables par les différents processeurs, nous proposons de nous appuyer sur une hiérarchie mémoire composée de mémoires lignes et de registres de voisinages. Ces derniers sont directement connectés aux processeurs de calcul qui peuvent les lire pendant la durée d'exécution d'un traitement. Comme pour la répartition des processeurs de calcul au sein des tuiles, les mémoires ligne sont des ressources limitées pour lesquelles une organisation optimale ou flexible devra être déterminée.

3.2.4 Contrôle des processeurs

Les processeurs *SplitWay* sont regroupés en mode Multi-SIMD au sein de tuiles de calcul destinées à exécuter les différents traitements. De nouvelles données à traiter leur sont transmises au fur et à mesure de l'arrivée du flux de données par le gestionnaire de voisinages, tous les Nb_{Procs} pixels. Le contrôle des processeurs est réalisé au niveau d'une tuile de calcul par une « unité de contrôle ».

La fonction de cette unité de contrôle est de gérer les signaux qui permettent de démarrer un traitement, transmettre les différentes instructions aux processeurs concernés, transférer le ou les résultats de ce traitement vers la sortie. Sa principale tâche consiste donc à identifier quel est le segment de code à envoyer, à quels processeurs, et dans quels cas.

3.2.4.1 Segments de code disponibles

Plusieurs segments de code sont disponibles, chacun correspondant à l'exécution d'une partie différente d'un programme. Les flux de données issus de capteurs vidéo sont généralement constitués d'un délai d'inter-trame, de quelques dizaines de lignes, d'un signal de fin de ligne qui correspond à quelques dizaines de pixels et enfin de l'image qui peut être brute, en niveaux de gris, ou en couleurs. Différentes phases de traitement de l'image permettent d'exécuter différents segments de codes comme ceux définis comme suit :

- Un segment de code d'inter-trame permet d'exécuter des traitements entre chaque image. Comme ce délai dure l'équivalent de plusieurs lignes, des programmes de taille conséquente peuvent être lancés, typiquement pour effectuer une normalisation d'histogramme, échanger des données etc.
- Un segment de code de fin de ligne, qui dure l'équivalent de quelques dizaines de pixels, permet de réaliser des recherches des valeurs mininales ou maximales sur les pixels d'une même ligne, calculer des statistiques qui seront utilisées par la suite etc.
- Enfin, plusieurs segments de codes correspondent aux différents cas Multi-SIMD à exécuter. Il s'agit par exemple des traitements à exécuter en fonction de la position du pixel sur le filtre de couleur placé devant le capteur, ou encore de cas d'exécutions précalculées permettant d'implémenter efficacement les structures de type `IF` multiples et `CASE`.

3.2.4.2 Détermination du segment de code à exécuter

La tuile de calcul peut fonctionner en mode SIMD ou Multi-SIMD comme décrit précédemment. En mode Multi-SIMD, le choix du segment de code à exécuter est indiqué par la valeur d'une métadonnée. Lorsqu'il s'agit d'images brutes, cette donnée est transmise automatiquement à l'unité de contrôle. Le programmeur décrit alors autant de segments de code qu'il y a de cas possibles[3]. Ce mode de fonctionnement est adapté lorsque des traitements sont décrits avec des prédicats `CASE` qui rendent l'exécution du code irrégulière, ainsi

3. Un parallèle peut être fait avec un gestionnaire d'interruptions si l'on considére qu'à chaque nouveau pixel à traiter une interruption démarre le segment de code lui correspondant.

que pour le traitement d'images brutes. Lorsque le mode de fonctionnement choisi est le mode SIMD, le même segment est transmis séquentiellement à tous les processeurs d'une même tuile de calcul.

Toutes les informations déterminant le segment de code à lire et à transmettre par les différents processeurs *SplitWay* sont connues par le gestionnaire de voisinages qui a accès aux valeurs des données à traiter. Il lui suffit de les transmettre à l'unité de contrôle en fonction des paramètres de configuration de la tuile de calcul.

Conclusion sur le contrôle des processeurs

Nous proposons d'utiliser une unité de contrôle pour chaque tuile de calcul. Celle-ci permet de distribuer les instructions en mode SIMD et Multi-SIMD aux différents processeurs qui la composent. En association avec le gestionnaire de voisinages, elle est capable de décharger le programmeur de toutes les tâches de parcours de l'image. Des fonctions gèrent de manière transparente les prédicats `IF` et `CASE`, permettant ainsi le portage de traitements complexes et irréguliers sur une telle structure.

3.2.5 Détermination de l'interconnexion entre les éléments de l'architecture

Le modèle architectural comporte plusieurs dizaines de processeurs *SplitWay* et des éléments de mémorisation associés à des gestionnaires de voisinages qui se chargent d'apporter les données aux processeurs. Le nombre de ces éléments dépend essentiellement du budget en surface silicium et en consommation électrique. Ces composants sont organisés en tuiles de calcul intégrant les ressources nécessaires au contrôle et à l'accès aux données. Elles peuvent être enchaînées les unes aux autres pour constituer l'architecture de calcul dans son ensemble.

3.2.5.1 Reconfigurabilité gros grain

Nous avons vu que les ressources requises pour l'exécution d'un traitement varient d'un algorithme à l'autre. Une structure définie au pire cas implique une surface silicium et une consommation électrique telle qu'elle devient inexploitable dans le domaine de l'embarqué. Il est donc possible de figer la structure de calcul dès la conception du circuit pour répondre aux besoins d'une application donnée, ou bien de la rendre reconfigurable afin d'exploiter au mieux les ressources de calcul en fonction des chaînes de traitements qui sont portées.

C'est pourquoi, nous proposons d'étudier comment il est possible de reconfigurer les chemins d'interconnexion entre les différents éléments de l'architecture en fonction des besoins applicatifs. L'arrangement optimal des ressources peut être réalisé par le programmeur ou par des outils logiciels spécifiques à l'architecture dès l'écriture des algorithmes. Par exemple, les bancs mémoires inutilisés pour mémoriser des lignes de données peuvent être exploités en tant que mémoire de travail, et le regroupement de processeurs *SplitWay* au sein de tuiles de calcul peut être ajusté en fonction des algorithmes à exécuter. Une telle flexibilité nécessite que tous les éléments qui composent l'architecture puissent communiquer ensemble.

Structure d'interconnexion De nombreuses techniques sont présentées dans la littérature permettant d'interconnecter différents éléments d'un même composant entre eux [Raghunathan 2003]. Dans notre cas, les éléments déterminant le nombre de tuiles de calcul maximum sont les gestionnaires de voisinages et les unités de contrôle. Ainsi, si Nb_{UC} unités de contrôles et Nb_{Gv} gestionnaires de voisinages sont disponibles, alors le nombre maximal de tuile de calcul est de Nb_{UC}. Les autres ressources peuvent ensuite être connectées à ces composants ou désactivées afin de limiter leur consommation électrique.

Si Nb_{Procs} processeurs doivent être interconnectés à Nb_{UC} unités de contrôle, chacune leur transmettant des mots d'instructions VLIW, alors Nb_{Procs} démultiplexeurs 1 vers Nb_{UC} doivent être utilisés pour la seule transmission des instructions aux processeurs. Autant de multiplexeurs sont nécessaires pour transmettre les éventuels registres d'état des processeurs vers les unités de contrôle. De même, nous avons vu que les processeurs communiquent avec leurs plus proches voisins, il est donc nécessaire d'intégrer $1 \times Nb_{Procs}$ multiplexeurs Nb_{Procs} vers 1 (un pour chaque voisin *est* et *ouest*).

Comme la taille du voisinage se veut configurable, les mémoires ligne et les registres du gestionnaire de voisinages doivent être interconnectés entre eux. Là encore, les ressources nécessaires pour interconnecter les bancs mémoire disponibles aux gestionnaires de voisinages se révèlent importantes. De plus, ces bancs mémoire peuvent être potentiellement utiles par les processeurs en tant que mémoire de travail s'ils sont libres etc.

Bien que l'interconnexion des différents éléments puisse être hiérarchisée et qu'il ne s'agisse pas d'un réseau pleinement connecté, les ressources nécessaires peuvent se révéler particulièrement importantes si un haut niveau de reconfigurabilité est recherché. Cette solution reste toutefois envisageable pour une implémentation peu contrainte en surface et en consommation.

De plus, il a été vu précédemment qu'il était envisagé de sélectionner la fréquence de fonctionnement des processeurs *SplitWay* appartenant à une même tuile de calcul afin d'obtenir un compromis entre ressources de calcul disponibles et consommation électrique. La technique pressentie pour implémenter une telle fonctionnalité implique de disposer de plusieurs domaines et arbres d'horloges. Ces derniers sont configurés à l'initialisation des tuiles de calcul en fonction du traitement qui est porté. Ce degré de reconfiguration supplémentaire augmente d'autant la complexité globale du système.

Autres modes d'interconnexion D'autres modes de connexion peuvent être utilisés comme les Network On Chip (NoC) [Agarwal 2009, Hoi-Jun Yoo 2008, Hansson 2007] ou les bus [Salminen 2002]. La littérature permet de voir que les circuits utilisant les NoC souffrent d'un important surcoût en surface [Salminen 2008], pouvant parfois atteindre la moitié de la surface du composant. Ce surcoût représente parfois plusieurs dizaines de milliers de portes logiques [Arteris Co. 2005]. Les accès sont réalisés séquentiellement, une congestion peut donc apparaître lorsque leur nombre est important, ce qui limite la *scalabilité* du système. De plus, la plupart de ces solutions introduisent une latence de plusieurs cycles dans les communications, ce qui les rend inapplicables dans la plupart des cas qui nous concernent. Par conséquent, rendre la structure pleinement reconfigurable tout en maintenant un niveau élevé de performances et une surface silicium faible, est difficilement envisageable. C'est pourquoi il est prévu de limiter le niveau de reconfigurabilité de l'architecture afin de limiter sa surface et sa consommation.

Choix des éléments configurables Nous proposons de fixer le nombre de processeurs *SplitWay* intégrés à chaque tuile de calcul dès la conception du circuit, ce qui permet de leur associer systématiquement une unité de contrôle et un gestionnaire de voisinages auquel des éléments de mémorisation sont adjoints. La taille des voisinages peut être configurée entre 1×1 et $N_{Max} \times M_{Max}$, N_{Max} étant le nombre maximum de lignes qu'il est possible de mémoriser.

Le mode de fonctionnement SIMD ou Multi-SIMD est configuré pour chaque tuile de calcul ainsi que le format des données d'entrée et de sortie. La fréquence de fonctionnement peut être sélectionnée pour chaque tuile de calcul indépendamment les unes des autres, l'ensemble étant synchronisé par une horloge commune.

Ainsi, chaque tuile de calcul est autonome, ce qui rend le système entièrement *scalable*. Elles doivent pouvoir s'enchaîner les unes avec les autres en fonction des besoins de l'application.

3.2.5.2 Enchaînement des tuiles de calcul

Comme les tuiles peuvent être hétérogènes, tant par leurs ressources de calcul que de mémorisation, les outils de programmation doivent pouvoir porter les algorithmes sur celles les plus à même de les recevoir afin de limiter la consommation électrique. Pour cela, toutes les tuiles de calcul doivent pouvoir transmettre leur résultat à n'importe quelle autre tuile de calcul. Chaque tuile de calcul doit pouvoir recevoir et transmettre au moins une donnée à chaque cycle pixel. Des modules permettant de former des flux composites peuvent être ajoutés sur le bus.

Définition du bus et son protocole Aucune tuile de calcul n'a de fonction maître ou esclave, aussi avons nous choisi un contrôle distribué, chaque tuile assurant la lecture des données qui la concerne ainsi que leur écriture. Pour supprimer tout problème de congestion ou de collision, et assurer le fonctionnement en temps réel du bus, nous avons défini un protocole simple et flexible fonctionnant en TDMA.

Les tuiles de calcul se voient attribuées un identifiant unique lors de leur configuration ou de leur conception. Elles sont reliées à l'horloge du bus qui permet de le cadencer mais aussi de synchroniser les tuiles entre elles. Elles sont aussi reliées à au moins un canal destiné à transférer les données entre les tuiles. Ainsi la fréquence de fonctionnement du bus Bus_{Clk} est au plus celle de la tuile de calcul la plus lente.

Grâce à l'équation 3.8, il est ensuite possible de déterminer le nombre de divisions temporelles – que nous appellerons « slot »– disponibles pour la transmission d'un pixel. Ce nombre est à corréler au nombre de tuiles Nb_{Tuiles} devant communiquer entre elles, et de canaux du bus Nb_{Canaux} auxquels elles sont connectées. Il est dès lors possible de configurer chaque tuile pour qu'elle puisse lire la valeur des données qui lui sont destinées. Cette lecture a lieu à intervalles réguliers déterminés par l'horloge pixel. Cette configuration consiste à transmettre à la tuile de calcul lors de son initialisation le numéro du canal et du slot qui lui est attribué. Plusieurs tuiles de calcul peuvent lire simultanément le même slot, ce qui permet de réaliser des traitements différents sur un même jeu de données, composites ou non. De même, il suffit de configurer chaque tuile pour qu'elle puisse écrire ses résultats sur un slot prédéfini. Dans ce cas, seule une donnée peut être écrite par slot et par canal.

$$Nb_{Slots} = \left\lfloor \frac{Bus_{Clk}}{Pixel_{Clk}} \right\rfloor \tag{3.8}$$

Conclusion sur l'interconnexion

Les ressources que nous proposons d'intégrer à l'architecture de calcul sont multiples. Il s'agit essentiellement de processeurs de calcul *SplitWay*, de gestionnaires de voisinages et d'éléments de mémorisation associés à des unités de contrôle. La répartition optimale des différents éléments (processeurs, mémoires etc.) au sein de ces tuiles de calcul en fonction de la surface silicium disponible et des performances de calcul recherchées est un problème complexe.

Aussi, avons nous envisagé de mettre en œuvre et de rendre configurable l'interconnexion de l'ensemble des composants pleinement connectée ce qui permet de disposer d'une flexibilité maximale. Toutefois, une brève étude montre que la surface silicium d'une telle solution s'annonce trop importante pour qu'elle puisse être retenue dans le cadre d'une implémentation embarquée de l'architecture. C'est pourquoi, nous avons choisi de

fixer certains éléments de l'architecture dès sa conception, comme par exemple, la fréquence de fonctionnement des tuiles de calcul, le nombre de processeurs qui y sont intégrés et la taille maximale des voisinages qu'elles sont capables de traiter. Comme ces paramètres rendent les tuiles de calcul potentiellement hétérogènes, leur enchaînement est rendu paramétrable par l'utilisation d'un bus TDMA spécialement conçu, qui permet aussi d'assurer leur synchronisation.

Conclusion

Après avoir réalisé l'étude algorithmique définissant des orientations architecturales dans la première section de ce chapitre, la seconde a proposé un modèle original d'une architecture de calcul programmable capable de délivrer suffisamment de capacité de calcul pour traiter des flux vidéo HD en temps réel, que nous appellerons *eISP*, pour embedded Image Signal Processor.

L'analyse algorithmique démontre que l'accès aux données requiert la majorité des ressources de calcul alors que les opérations réalisées sont similaires d'un traitement à l'autre. La partie contrôle est liée à l'accès aux données puisqu'elle consiste à démarrer les segments de programme correspondant à la donnée qui doit être traitée, mais aussi à gérer les parcours de l'image. C'est pourquoi nous avons choisi de séparer ces ressources de la partie calcul. Cette dernière requiert le maximum de flexibilité puisque c'est à ce niveau que de grandes différences apparaîssent entre les traitements que ce soit en termes d'opérateurs utilisés ou en termes de ressources de calcul. Plusieurs dizaines de GOPs s'avèrent nécessaires pour la partie calcul du traitement de flux vidéo haute définition HD 1080p.

L'analyse algorithmique montre enfin qu'il est possible d'exploiter le parallélisme au niveau instructions. Nous avons ainsi mis en évidence qu'un jeu d'instructions orthogonal permettait d'une part de réduire le nombre d'instructions nécessaires aux traitements d'image, mais aussi de réduire le nombre de voies de calcul nécessaires pour l'exécution d'un même traitement. Ainsi, deux voies de calcul s'avèrent suffisantes, et sont remplies à plus de 60 % en moyenne.

Afin d'exploiter toutes ces caractéristiques, l'architecture proposée consiste en l'utilisation de plusieurs tuiles de calcul capables de traiter des flux de données en toute autonomie. Elles sont connectées entre elles par un bus que nous avons conçu pour rendre leur enchaînement programmable. Chacune de ces tuiles est composée de groupes de processeurs de calcul *SplitWay* qui fonctionnent en mode Multi-SIMD. Ce mode de fonctionnement assure une exécution efficace des traitements d'image irréguliers (contenant des structures `CASE`).

Les données du flux vidéo sont automatiquement apportées aux processeurs de calcul sous la forme d'une matrice de registres contenant la valeur des pixels à traiter ainsi que celle de leurs voisinages. Ces données pouvant être composites, leur format est configuré pour l'entrée et la sortie de chaque tuile de calcul, ainsi le programmeur accède de manière transparente aux différentes composantes qui les constituent.

Le processeur *SplitWay*, qui compose et décompose les tuiles de calcul, est un processeur VLIW deux voies semi-orthogonal. Les opérateurs et les registres sont mutualisés entre les deux voies, ce qui permet de limiter la surface silicium, et simplifie la programmation en rendant symétrique l'utilisation des deux voies. De plus, le processeur *SplitWay* intègre un plan mémoire auquel peuvent être connectés LUTs, mémoires de travail ou accélérateurs dédiés.

Pour obtenir une capacité de calcul de l'ordre de 20 GOPs en respectant les contraintes de surface et de consommation, respectivement de quelques millimètres carrés pour quelques centaines de milliwatts, une cinquantaine de processeurs sont nécessaires. Il a été estimé dans ce chapitre que les processeurs à implémenter en technologie 65 nm, doivent présenter une consommation électrique inférieure à 7 milliwatts pour une surface de 0,03 mm^2 soit moins de 13,5 kGates.

CHAPITRE 4

Conception de l'architecture eISP

Où sont rappelés les principaux éléments de l'architecture de calcul eISP.
Où sont détaillés les choix d'implémentations et les éléments constitutifs de l'architecture.

Introduction

LE CHAPITRE précédent propose un modèle architectural basé sur l'enchaînement de tuiles de calcul. Celles-ci regroupent des processeurs programmables VLIW deux voies fonctionnant en mode Multi-SIMD, associés à des éléments de mémorisation, de contrôle et de communication. Nous avons vu que cette solution peut répondre aux contraintes de surface silicium et de consommation électrique imposées par le cahier des charges tout en offrant une capacité de calcul suffisante pour le traitement vidéo HD 1080p en temps réel. Ce chapitre présente la conception de l'architecture *eISP*.

Après un bref rappel des différentes caractéristiques de l'architecture *eISP* dans la première section, les sections suivantes de ce chapitre détaillent sa conception. Chaque tuile de calcul intègre un gestionnaire de voisinages dont le mécanisme est décrit en section 4.2. La conception du processeur *SplitWay* VLIW deux voies est détaillée en section 4.3. La technique mise en œuvre pour assurer la décomposition des données composites contenant des métadonnées, sans que cela ne soit à gérer par le programmeur, est détaillée en 4.4. L'unité de contrôle, qui consiste essentiellement à lire et à transmettre les instructions aux processeurs *SplitWay* est décrite en section 4.5. Les résultats de l'exécution des programmes sont ensuite envoyés à un module de communication qui les transmet à d'autres tuiles de calcul. Ce module fait l'objet de la section 4.6. Enfin, la section 4.7décrit comment l'architecture *eISP* a été conçue pour recevoir des ressources de calcul dédiées permettant d'augmenter sa capacité de calcul en utilisant l'infrastructure d'accès aux données et de calcul mise en place.

4.1 Description de l'architecture

Nous avons vu dans le chapitre précédent que le fonctionnement des processeurs en mode Multi-SIMD était le plus à même d'exploiter le parallélisme spatial inhérent au traitement d'image. Cette section a pour but de rappeler brièvement les principales caractéristiques du modèle architectural *eISP* qui ont été définies dans le chapitre précédent à partir de l'analyse algorithmique. La première partie de cette section est consacrée à une description globale de l'architecture *eISP*. Celle-ci est organisée en tuiles de calcul programmables dont le principe de fonctionnement est décrit dans la seconde partie.

4.1.1 Description globale de l'architecture

L'architecture *eISP* peut être vue comme un ensemble de tuiles de calcul programmables interconnectées entre elles et capables de traiter des flux de données. Les tuiles de calcul sont autonomes et peuvent être remplacées par des PIs dédiées lorsque cela se révèle nécessaire. La Figure 4.1 schématise la structure globale de l'architecture *eISP*.

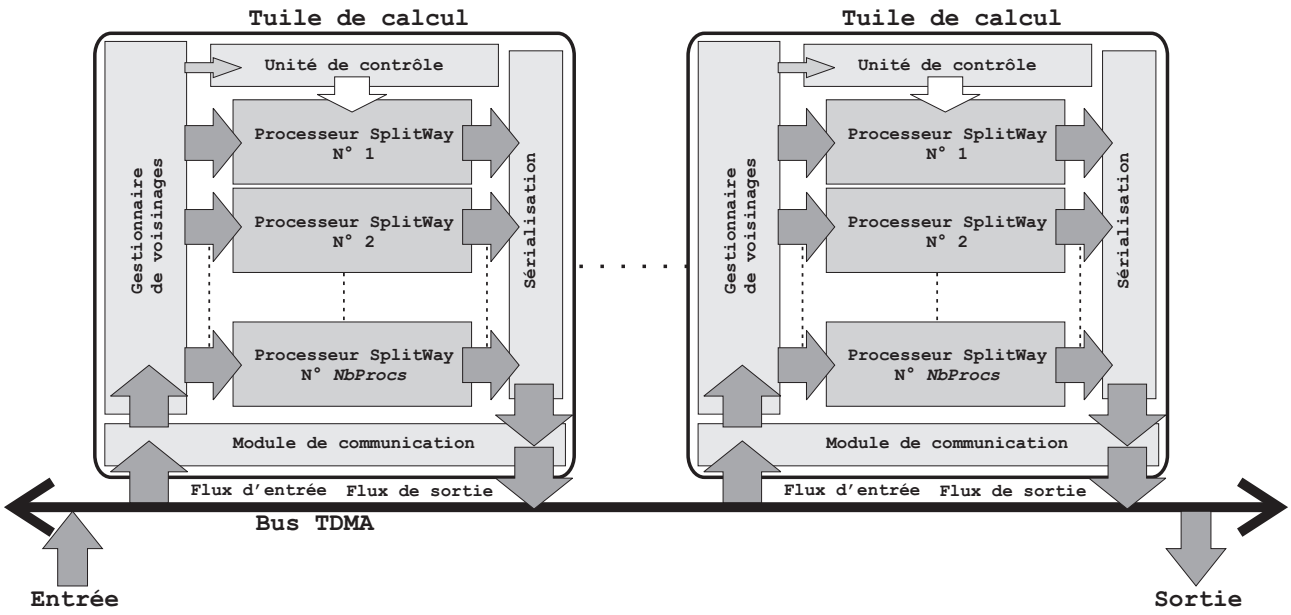


FIG. 4.1: EXEMPLE DE TUILES INTERCONNECTÉES ENTRE ELLES PAR LE BUS TDMA.

Les tuiles sont interconnectées entre elles par un bus qui permet de les alimenter en données, mais aussi de les maintenir synchronisées grâce à son horloge propre. La principale vocation de ce bus est de permettre de définir l'ordre d'enchaînement des tuiles de calcul par simple écriture de la valeur d'un registre d'adresses. Les données qui arrivent en flux depuis ce bus peuvent être des mots de données composites, intégrant des métadonnées. C'est aussi le cas pour les données sortantes. Un mécanisme permet de configurer le format de ces données au niveau de chaque tuile de calcul rendant l'extraction des différentes parties des mots transparente et instantanée pour le programmeur.

Ce mode de fonctionnement en flux de données permet d'exploiter le parallélisme inhérent à la structure *pipelinée* des chaînes algorithmiques de traitement d'image. Ainsi l'architecture *eISP* est *scalable* grâce à l'indépendance de ses tuiles de calcul. Leurs ressources de calcul programmables sont obtenues par l'utilisation de processeurs VLIW deux voies spécialement conçus.

4.1.2 Fonctionnement des tuiles de calcul

Les tuiles de calcul sont constituées de Nb_{Procs} processeurs programmables *SplitWay*. Ceux-ci fonctionnent en mode Multi-SIMD ce qui permet d'exploiter au mieux les ressources de calcul, notamment lorsque des traitements irréguliers doivent être portés sur l'architecture tout en bénéficiant des avantages du mode SIMD en flux de données. Dans notre cas, ce mode de fonctionnement nous permet d'obtenir le nombre de cycles disponibles par pixel pour l'exécution d'un traitement, grâce à la fréquence de fonctionnement des processeurs $Proc_{Clk}$ et la fréquence du flux de données $Pixel_{Clk}$, donné par l'équation 4.1.

$$NbCycles_{Disponibles} = \left\lfloor \frac{Proc_{Clk}}{Pixel_{Clk}} \times Nb_{Procs} \right\rfloor \tag{4.1}$$

Ainsi, les tuiles de calcul peuvent exécuter de manière optimale des programmes utilisant des prédicats de type `CASE`, ce qui n'est pas le cas avec des structures SIMD traditionnelles où on observe un surcoût important en temps de traitement.

Il est crucial de maximiser le nombre d'opérations exécutées par cycle d'horloge, c'est pourquoi différentes formes de parallélisme sont exploitées. Le parallélisme de tâche est réalisé par l'enchaînement des traitements portés sur les tuiles de calcul mais aussi en réalisant l'ensemble des opérations d'accès aux données, de contrôle et de communication en temps masqué. De plus, le parallélisme de données est exploité grâce au fonctionnement Multi-SIMD tandis que le parallélisme d'instruction l'est au niveau des processeurs de calcul, grâce aux deux voies de calcul.

4.1.2.1 Le mode d'accès aux données

La plupart des algorithmes recensés nécessitent l'accès aux valeurs des pixels d'un voisinage autour du pixel à traiter, comme c'est par exemple le cas pour une convolution. Le gestionnaire de voisinages permet d'apporter les données à traiter directement aux différents processeurs de la structure. Pour cela, des registres de voisinages, dont la taille est configurable, mettent à disposition des processeurs de calcul l'ensemble des valeurs des pixels à traiter pendant toute la durée du traitement. Ceci est réalisé grâce à une hiérarchie mémoire préparant toutes les données en temps masqué.

4.1.2.2 Adaptation de la fréquence de fonctionnement

Comme les tuiles de calcul sont autonomes, elles peuvent fonctionner à différentes fréquences d'horloges qui sont déterminées par les outils logiciels de l'architecture dès le portage des algorithmes. En effet, l'équation 4.1 montre qu'il est possible de faire varier la capacité de calcul en sélectionnant une fréquence d'horloge adaptée à la durée d'exécution des algorithmes, durée connue dès leur portage. La stratégie est d'utiliser la fréquence la plus faible disponible permettant d'assurer l'exécution des traitements en temps réel et ainsi minimiser la consommation électrique.

4.1.2.3 Utilisation de processeurs VLIW deux voies

Nous avons choisi d'intégrer deux voies de calcul au processeur *SplitWay* pour exploiter au mieux le parallélisme d'instruction, ce qui permet de réduire la durée d'exécution des traitements comme nous l'avons vu dans le chapitre précédent. Les opérateurs et la file de registres sont mutualisés pour les deux voies de calcul de manière à limiter la surface silicium utilisée. L'accès aux opérandes est orthogonal afin de réduire les opérations utilisées pour le préchargement des données en file de registres.

4.1.2.4 Gestion des métadonnées

Les données manipulées par les tuiles de calcul peuvent être composées de plusieurs pixels ou intégrer des métadonnées (résultats de calcul précédents etc.) dont le format varie en fonction des traitements. C'est pourquoi nous avons rendu configurable leur format pour les entrées et les sorties de chaque tuile de calcul. De plus un mécanisme transparent pour le programmeur permet l'extraction des différentes parties de ces mots. Son utilisation est similaire à celle de registres sécables.

4.1.2.5 L'unité de contrôle

L'unité de contrôle permet de démarrer les segments de code en fonction des traitements à exécuter et de les distribuer aux différents processeurs d'une même tuile de calcul. Elle permet de contrôler le mode de fonctionnement Multi-SIMD sur les métadonnées. Ainsi un segment de code différent peut être transmis aux processeurs concernés en fonction de la valeur que prend un résultat de calcul obtenu sur une tuile de calcul précédente, ou encore de la position d'un pixel dans l'image brute.

4.1.2.6 Le module de communication

Ce module permet de lire et écrire les données traitées sur le bus TDMA qui assure les communications sous forme de flux de pixels entre les tuiles de calcul. Il a été conçu afin de rendre configurable l'enchaînement des différentes tuiles de calcul sur lesquelles des traitements sont programmés. De plus, comme les tuiles de calcul peuvent fonctionner à des fréquences de fonctionnement différentes, l'horloge du bus TDMA permet de maintenir l'ensemble synchronisé.

Conclusion sur les rappels relatifs à l'architecture

Cette première section a permis de rappeler brièvement l'organisation de l'architecture *eISP*, des ses principales caractéristiques ainsi que des éléments qui la composent. La section suivante détaille l'implémentation et le fonctionnement de l'architecture *eISP*.

4.2 Implémentation du gestionnaire de voisinages

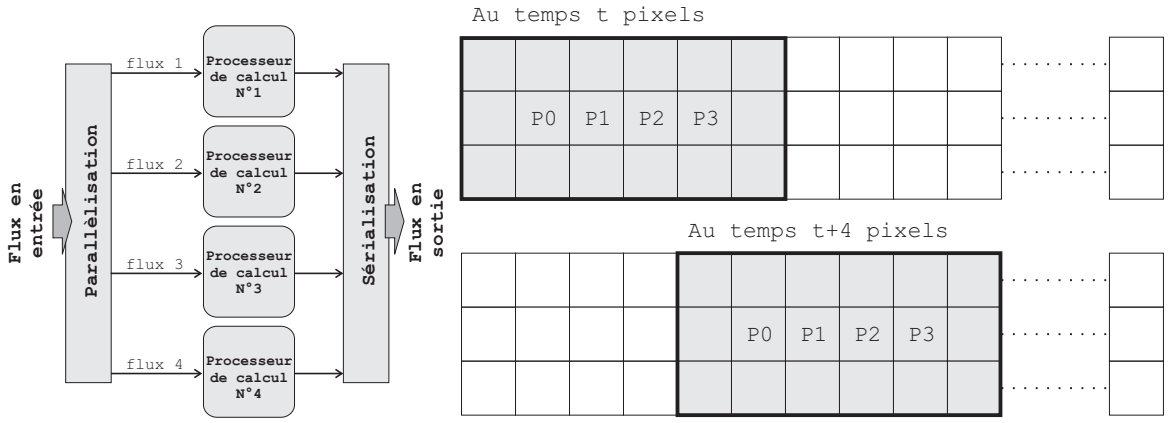
Nous avons vu dans le chapitre 3 que près de la moitié des instructions d'un programme est affectée à cette tâche. Les données à traiter sont transmises en série au gestionnaire de voisinages qui a pour fonction de les organiser afin que les différents processeurs puissent les lire sous forme de matrices. Ainsi, chacun d'entre eux peut accéder directement aux valeurs des pixels ainsi qu'à leurs voisinages sans avoir à supporter le coût du calcul d'adresses. Autrement dit, il s'agit de donner aux processeurs un accès au pixel à traiter ainsi qu'à ses voisins.

Afin de permettre une utilisation flexible de la tuile de calcul, le gestionnaire de voisinages peut être configuré par le programmeur pour qu'il gère des voisinages de taille $N \times M$, où N est le nombre de lignes du voisinage ($N <= N_{max}$) et M le nombre de colonnes du voisinage, ($M <= M_{max}$). Les processeurs peuvent donc accéder à des voisinages de 1×1 pixel jusqu'à $N_{max} \times M_{max}$ pixels.

4.2.1 Mécanisme de mémorisation des données

Comme nous l'avons évoqué précédemment les données sont traitées simultanément par les Nb_{Procs} processeurs *SplitWay* d'une même tuile de calcul. Comme les données arrivent en flux, le gestionnaire de voisinages parallélise ces flux afin de les distribuer aux processeurs. Les Figures 4.2a et 4.2b présentent l'exemple d'un flux parallélisé sur 4 processeurs ayant chacun l'accès à un voisinage de taille 3×3 pixels. La fonction ainsi réalisée équivaut à une fenêtre glissante qui se déplace de Nb_{Procs} pixels, couvrant à chaque fois un segment de $Nb_{Procs} + M - 1$ pixels. Comme les pixels d'un même segment sont contigus, il y a recouvrement de leurs voisinages comme l'illustre la Figure 4.7. C'est pourquoi les registres de voisinages sont mutualisés pour l'ensemble des processeurs, réduisant ainsi leur nombre.

Il y a potentiellement plus de processeurs qui cherchent à lire des valeurs de voisinages simultanément qu'il n'y a de ports aux bancs mémoire utilisés. L'utilisation de mémoire à ports multiples ou la mémorisation des lignes dans des registres étant des solutions trop coûteuses en surface silicium et en consommation électrique, nous avons proposé une hiérarchie mémoire qui consiste à mémoriser les lignes dans des bancs simple port et de maintenir les valeurs des registres de voisinages à jour pour que les accès aux données puissent être concurrents.



(a) PARALLÈLISATION D'UN FLUX DE DONNÉES POUR 4 PROCESSEURS.

(b) ACCÈS À L'IMAGE PAR SEGMENT DE 4 PIXELS CONSÉCUTIFS.

FIG. 4.2: EXEMPLE DE DISTRIBUTION D'UN FLUX DE DONNÉES SUR 4 PROCESSEURS.

4.2.1.1 Utilisation de mémoires ligne

Afin de constituer autant de voisinages de taille $N \times M$ qu'il y a de processeurs (Nb_{Procs}), nous avons besoin de N mémoires ligne dans lesquelles mémoriser les données au fur et à mesure de leur arrivée. Pour des raisons de surface silicium et de consommation d'énergie, nous privilégierons des mémoires Random Access Memory (RAM) simple port aux mémoires de type FiFo, utilisant des mémoires double port, technologies classiquement usitées dans les architectures de traitement d'image.

La consommation électrique dynamique des bancs mémoire est directement corrélée au nombre d'accès réalisés par seconde, nous chercherons donc à limiter ces accès. Par ailleurs, la surface silicium d'un banc mémoire est liée au nombre de ports de lecture et d'écriture. Nous avons donc mis en place un mécanisme qui permet d'exploiter des mémoires simple port, composant standard proposé par les fondeurs, et d'en limiter les accès tant en lecture qu'en écriture, quel que soit le nombre de processeurs utilisés ou leur activité.

4.2.1.2 Mise en forme des voisinages

Pour que les valeurs des voisinages soient en permanence accessibles par les processeurs, leurs valeurs sont stockées dans des registres. Ils apportent les $N \times M$ valeurs du voisinage correspondant aux Nb_{Procs} processeurs afin qu'ils puissent les lire pendant toute la durée du traitement. Lorsqu'une donnée entrante (un pixel) est reçue par le gestionnaire de voisinages, le processus décrit par l'algorithme de la Figure 4.3a et par les Figures 4.5 et 4.6 est mis en œuvre pour recopier la valeur du nouveau pixel dans les mémoires ligne à la bonne position. Des registres tampons sont utilisés pour remplir les registres de voisinages à partir des mémoires ligne tout en maintenant les valeurs des registres de voisinages stables, comme en Figure 4.4, alors que le flux de pixel continue à entrer. Ils servent aussi à réordonner les valeurs des pixels lorsque cela est nécessaire, comme c'est par exemple le cas en 4.6, en se basant sur l'algorithme de la Figure 4.3b. C'est ainsi les Nb_{Procs} processeurs ont accès aux registres de voisinages correspondant à un segment de Nb_{Procs} successifs.

On voit qu'un seul accès en écriture est réalisé sur une des mémoires ligne, et qu'un accès en lecture est réalisé sur chacune des autres à l'arrivée de chaque nouveau pixel du flux (c'est à dire à la fréquence de l'horloge pixel). Il est donc possible d'utiliser des mémoires simple port à la fréquence de l'horloge pixel, très inférieure à celle des processeurs, ce qui maintient la consommation sous contrôle. La machine à état du

```
1: Lc ← 0
2: Li ← 0
3: imW // Largeur de l'image en pixel
4: PixelEntrant // valeur du pixel entrant
5: if Lc < imW then
6:     Lc ← Lc + 1
7: else
8:     Lc ← 0
9:     if Li > N then
10:        Li ← 0
11:    else
12:        Li ← Li + 1
13:    end if
14: end if
15: Mem[Li][Lc] ← PixelEntrant
```

(a) ALGORITHME DE MISE À JOUR DES MÉMOIRES LIGNE.

```
1: Lc // écrit par l'algorithme précédent
2: Li // écrit par l'algorithme précédent
3: PixelEntrant // valeur du pixel entrant
4: for i ← 0 to M − 1 do
5:     Voisinage[i] = (Li + i + 1)%N
6: end for
7: Voisinage[M − 1] ← PixelEntrant
```

(b) ALGORITHME DE MISE À JOUR DES REGISTRES TAMPONS À PARTIR DES MÉMOIRES LIGNES.

FIG. 4.3: ALGORITHMES UTILISÉS PAR LES MACHINES À ÉTAT DES GESTIONNAIRES DE VOISINAGES (a) POUR LA MISE À JOUR DES MÉMOIRES LIGNE, (b) POUR LA MISE À JOUR DES REGISTRES DE TAMPONS.

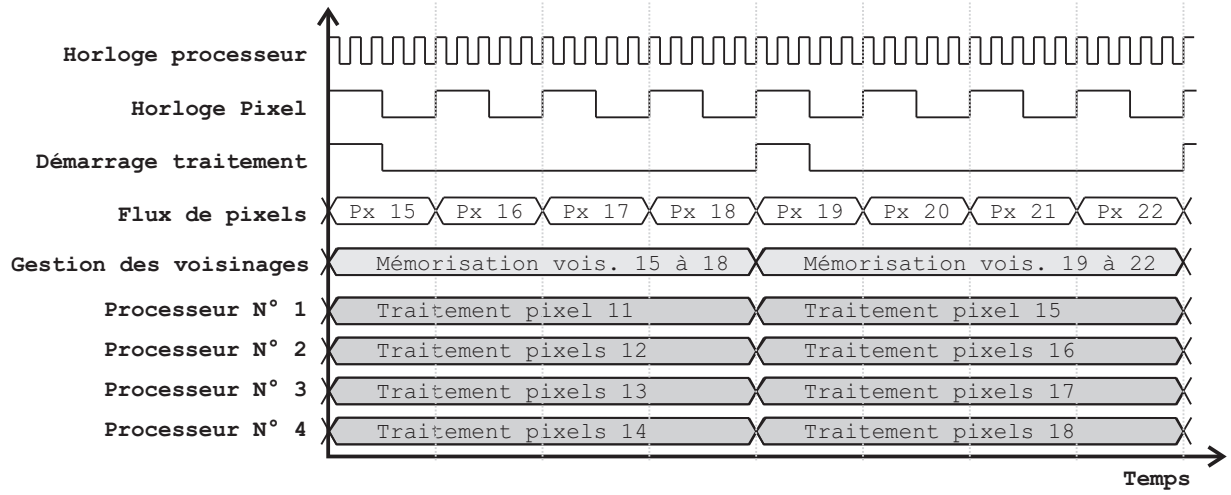


FIG. 4.4: EXEMPLE DE CHRONOGRAMME DE L'ACCÈS AUX VOISINAGES POUR UNE TUILE DE 4 PROCESSEURS.

gestionnaire de voisinages consiste en la gestion de quelques compteurs, de recopies et décalages de valeurs de pixels. La plus grande part des ressources est attribuée aux registres et aux mémoires ligne.

4.2.2 Performance, surface et consommation du gestionnaire de voisinages

La méthode décrite précédemment permet d'organiser les données afin de les rendre directement utilisables par les processeurs et supprime ainsi la plupart des instructions du programme, nécessaires au calcul d'adresse et à l'organisation de ces voisinages pendant l'exécution de l'algorithme. Comme les voisinages des pixels à traiter sont écrits dans $N \times M$ registres, le nombre de processeurs pouvant lire leurs valeurs n'est limité que par la sortance des transistors qui composent les registres. Bien que le gestionnaire de voisinages ne réalise pas de

calculs, sont rôle est crucial puisqu'il apporte aux processeurs les données sur lesquelles ils doivent réaliser les traitements. Aussi l'évaluation de ses performances est essentielle, tant en vitesse de lecture, qu'en surface ou consommation.

4.2.2.1 Débit en lecture

Sa performance est évaluée par le volume de données qu'il est capable d'apporter aux processeurs. En d'autres termes, il s'agit du nombre de pixels que l'ensemble des processeurs peuvent lire simultanément. Pour Nb_{Procs} processeurs fonctionnant à la fréquence $Proc_{Clk}$ possédant $N \times M$ ports de lecture de largeur $DataWidth$ bits, l'équation 4.2 donne le débit maximum en lecture des données du voisinage. Pour des processeurs ayant $NbPorts$ ports d'entrées sorties, le débit en lecture des données du gestionnaire de voisinages est déterminé par l'équation 4.3 avec $(NbPorts <= N \times M)$.

$$D_{Max} = N \times M \times Nb_{Procs} \times UcClk \times DataWidth \tag{4.2}$$

$$D_{nom} = NbPorts \times Nb_{Procs} \times UcClk \times DataWidth \tag{4.3}$$

4.2.2.2 Surface silicium

Le nombre de lignes N du voisinage est le paramètre prépondérant dans l'équation 4.4 qui exprime le nombre de registres Nb_{Regs} utilisés. Par ailleurs, l'équation 4.5 indique que le gain en nombre de registres tend vers M lorsque Nb_{Procs} tend vers l'infini par rapport à une version ne les mutualisant pas. En pratique, il est aisé d'obtenir des gains supérieurs à $\frac{M}{2}$ dès la mise en parallèle de plus de 4 unités de calcul.

$$Nb_{Regs} = (Nb_{Procs} + M - 1) \times N \tag{4.4}$$

$$\lim_{Nb_{Procs} \to \infty} \frac{Nb_{Procs} \times N \times M}{Nb_{Regs}} = M \tag{4.5}$$

4.2.2.3 Consommation électrique

La consommation électrique du gestionnaire de voisinages est obtenue par la somme des consommations des bancs mémoire utilisés pour stocker les lignes, des registres de voisinages et des registres tampons, ainsi que de la logique de contrôle associée, comme le présente l'équation 4.8. La consommation électrique des bancs mémoire est obtenue en sommant leurs consommations statiques et dynamiques comme le présente l'équation 4.6. Des bancs mémoire standards étant utilisés, il est possible de se baser sur les chiffres communiqués par les fondeurs pour leurs technologies d'intégration.

Comme cela a été décrit précédemment, à chaque nouveau pixel, une écriture est réalisée en mémoire tandis que $(N-1)$ lectures le sont sur les autres bancs. L'équation 4.7 permet de calculer cette consommation en considérant que les bancs mémoire sont homogènes. Par ailleurs, la consommation électrique de registres de voisinages et de la logique qui permet de les remplir à partir des mémoires ligne peut être déterminée par simulation du circuit en fonctionnement après synthèse sur les technologies d'intégration cibles.

$$Conso_{\text{Bancs Mémoire}} = \underbrace{\sum Conso_{Statique}}_{\text{Consommation statique en W/s}} + \underbrace{\sum Conso_{Lecture} + Conso_{Ecriture}}_{\text{consommation dynamique en W/s}} \tag{4.6}$$

$$Conso_{\text{MémoiresLignes}} = \underbrace{N \times Conso_{Statique}}_{\text{Consommation statique en W/s}} + \underbrace{(N-1) \times Conso_{Lecture} + 1 \times Conso_{Ecriture}}_{\text{Consommation dynamique en W/s}} \tag{4.7}$$

$$Conso_{GvoisN \times M} = \sum_{N \times (M+1+Nb_{Procs})} Conso_{Registre} + Conso_{\text{MémoiresLignes}} + Conso_{Logique} \tag{4.8}$$

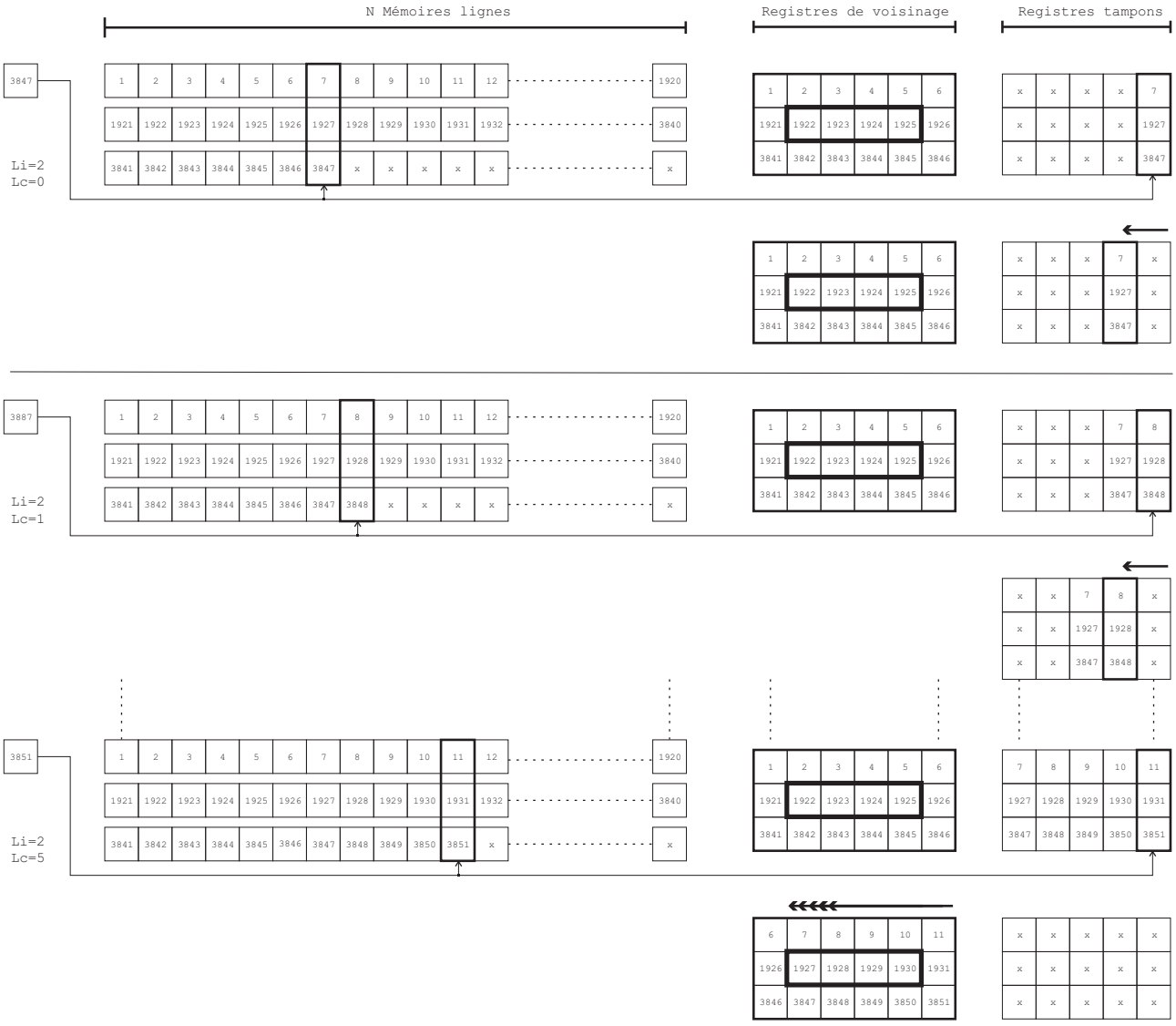


FIG. 4.5: SÉQUENCE DE RECONSTITUTION DES VOISINAGES POUR 4 PROCESSEURS À PARTIR DU FLUX DE PIXELS (DE HAUT EN BAS).

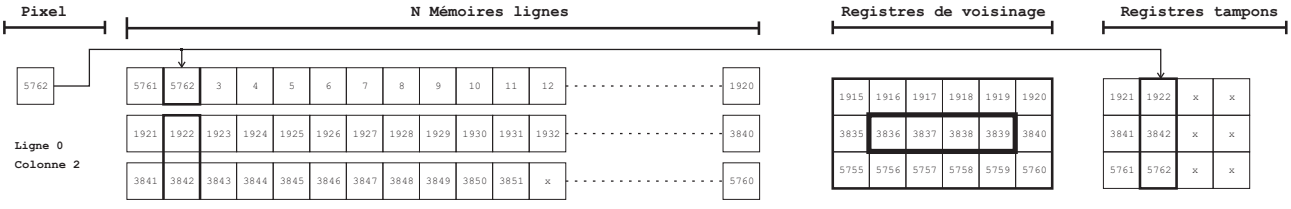


FIG. 4.6: RÉORGANISATION DES LIGNES LORS DE LA RECONSTRUCTION DES VOISINAGES POUR 4 PROCESSEURS.

Conclusion sur le gestionnaire de voisinages

Le gestionnaire de voisinages permet de réduire de plus de 50 % le nombre d'opérations d'un programme, c'est pourquoi il a fait l'objet de toute notre attention. Il est conçu pour gérer des tailles de voisinages que l'utilisateur peut configurer tout en limitant la surface silicium et la consommation électrique. En effet il est construit autour de bancs mémoire simple port pour lesquels les accès sont réalisés à la fréquence de l'horloge pixel.

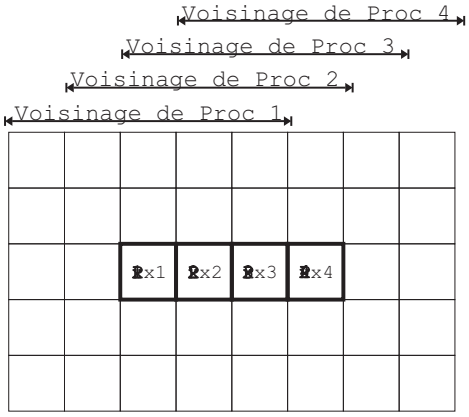


FIG. 4.7: LES REGISTRES DE 4 VOISINAGES 5×5 MUTUALISÉS POUR 4 PROCESSEURS.

Ces bancs mémoire sont lus pour reconstituer les valeurs des Nb_{Procs} voisinages que les Nb_{Procs} processeurs SplitWay doivent traiter dans des registres prévus à cet effet. Là encore, afin de limiter la surface silicium, le nombre de ces registres bénéficie du recouvrement des voisinages lié à consécutivité des Nb_{Procs} pixels comme le rappelle la Figure 4.7. De plus, seuls des accès en lecture sont autorisés ce qui réduit la surface silicium liée à la gestion des accès multiples aux valeurs.

Toutes les valeurs des voisinages du segment de pixels en cours de traitement peuvent donc être lues simultanément par les processeurs, mais aussi par d'autres composants de la tuile de calcul si cela est nécessaire.

4.3 Implémentation du processeur de calcul

Nous avons vu que le flux de pixels à traiter est mis à disposition des processeurs par les registres de voisinages, qui reconstituent le masque de voisinage autour du pixel que chaque processeur doit traiter. Pour assurer une capacité de calcul élevée tout en limitant la surface silicium utilisée et la consommation électrique, nous avons conçu le processeur *SplitWay* VLIW deux voies. En effet, nous avons vu dans le chapitre 3, qu'il était possible d'exprimer efficacement les programmes sur deux niveaux de parallélisme d'instruction.

De plus, le processeur *SplitWay* intègre deux voies de calcul orthogonales offrant un accès direct aux valeurs des opérandes, ce mode d'accès permet de réduire les opérations utilisées pour les chargements de données en file de registres. L'implémentation des opérateurs identifiés comme nécessaires dans le chapitre précédent, est d'abord présentée. Le jeu d'instructions que nous avons conçu pour exploiter ces opérateurs est ensuite détaillé, suivi de l'implémentation du *pipeline* complet du processeur.

4.3.1 Choix des opérateurs

Nous avons sélectionné les opérateurs à instancier dans le processeur *SplitWay* à partir de l'approche développée dans le chapitre 3. Elle consiste en une analyse dynamique du taux de remplissage des voies d'un processeur VLIW en fonction de leur nombre. L'étude est réalisée avec un jeu d'instructions orthogonal et un jeu d'instructions traditionnel.

Cette étude nous conduit à l'utilisation d'un processeur VLIW deux voies avec un jeu d'instructions partiellement orthogonal, ce qui signifie que pour chacune des voies, un des deux opérandes utilisables peut être issu de n'importe quelle source. Le second opérande doit être issu de la file de registres. Ce choix permet de

compacter la taille des instructions sans affecter notablement les performances, puisque seule une opération sur 14 utilise des opérandes qui ne sont pas stockés en file de registres, soit 7,1 % des instructions.

Sélection des opérateurs Afin de déterminer le jeu d'opérateurs, nous avons décrit les algorithmes représentatifs de la chaîne d'acquisition et d'amélioration d'image avec un jeu d'instructions réduit séquentiel en tenant compte des spécificités du modèle SIMD. Nous avons ainsi pu exécuter les algorithmes et obtenir la répartition des opérations utilisées lorsqu'un jeu d'instructions orthogonal est utilisé, dont un résumé est présenté en Table 4.1.

Ces résultats montrent que les opérations arithmétiques (additions et soustractions) représentent la grande majorité des opérations. Notons qu'elles sont aussi utilisées pour réaliser les comparaisons sur lesquelles se basent les instructions conditionnelles. Les multiplications, bien que souvent utilisées de manière intensive par certains algorithmes de traitement d'image, ne représentent au total que 14 % de l'ensemble des instructions appelées.

Par ailleurs, l'étude présentée dans le chapitre précédent a montré qu'il est possible de paralléliser efficacement les algorithmes sur deux voies. Ce qui signifie que de nombreuses opérations peuvent être exécutées simultanément. Il s'agit le plus souvent d'opérations différentes, mais pour déterminer l'organisation des opérateurs au niveau des voies du processeur VLIW, il est essentiel de connaître les opérateurs potentiellement appelés simultanément.

L'analyse du graphe d'exécution donne cette information. Ainsi, lorsqu'une addition ou une soustraction est exécutée sur la première voie, il y a 59 % de chance qu'une addition ou une soustraction soit utilisée sur la seconde voie. Comme ces opérations représentent plus de la moitié des instructions exécutées, il est nécessaire d'intégrer deux instances de ces opérateurs.

Les opérations de chargement et de stockage de données dans la file de registres ont une probabilité de 78 % d'être appelées simultanément. Cette forte probabilité, rapportée au 12 % de leur représentativité ainsi que la simplicité architecturale de l'opérateur, nous pousse à en intégrer deux instances. Lorsqu'une multiplication est exécutée sur une voie, la seconde voie n'a qu'une probabilité de 19 % d'exécuter elle aussi une multiplication. C'est pourquoi nous n'intégrons qu'un seul multiplieur au processeur.

Notons toutefois que ce qui est vrai pour le multiplieur est faux pour les Multiplications-ACcumulations (MACs) puisque si une voie en exécute une, la seconde voie à 60 % de chance d'exécuter elle aussi cette opération. Ceci s'explique par son intense utilisation au sein d'algorithmes réguliers. Aussi avons nous choisi de réaliser ces opérations en deux temps. Pour que l'utilisation de la Multiplication-ACcumulation (MAC) soit

TAB. 4.1: RÉPARTITION MOYENNE DES OPÉRATIONS POUR LES ALGORITHMES USUELS DE TRAITEMENT D'IMAGE LORSQU'UN JEU D'INSTRUCTIONS ORTHOGONAL EST UTILISÉ.

Opération entières signées	Représentation
Addition et soustraction	54 %
Multiplication	14 %
Accès aux données (chargement, copie entre registres...)	12%
Décalage	11 %
Logiques	7 %

rentable il est préférable d'en implémenter deux instances, ce qui est coûteux en termes de surface silicium. Bien que l'implémentation de MACs soit possible, nous nous limiterons, pour la version de base du processeur *SplitWay*, à n'intégrer qu'un seul multiplieur.

C'est pourquoi le processeur *SplitWay* intègre, dans sa version standard, les opérateurs entiers signés suivants:

- deux additionneurs/soustracteurs;
- deux unités d'affectation;
- un multiplieur;
- une unité de décalage signé pouvant aller jusqu'à 8 bits (shifter) ;
- une unité de stockage en mémoire externe.
- une unité booléenne (`AND` `OR` `NOT` `NAND` etc.)

Mutualisation des opérateurs Nous avons vu que certaines opérations sont souvent exécutées simultanément, comme les opérations de stockage, les additions et les soustractions. Traditionnellement, au sein des processeurs VLIW, le jeu d'opérateurs est dupliqué ou réparti entre les deux voies. La duplication de tous les opérateurs implique une flexibilité maximale au niveau de la programmation au détriment de la surface silicium. La répartition des opérateurs entre les voies permet de limiter la surface silicium utilisée au détriment de la flexibilité et nécessite un effort de programmation supplémentaire. La même politique est appliquée au niveau de la file de registres qui peut être unique et partagée, ou bien segmentée entre les voies.

Dans un souci de flexibilité, et pour assurer une utilisation élevée des deux voies du processeur, nous avons choisi de rendre symétrique l'accès aux ressources. Autrement dit, les différents opérateurs et les registres peuvent être appelés indifféremment sur les deux voies. Comme de nombreuses combinaisons d'opérateurs ne sont que très rarement exécutées simultanément, au lieu de dupliquer les ressources pour les deux voies, nous avons choisi de les mutualiser. Cela permet de réduire notablement la surface du module regroupant les opérateurs par rapport à une solution dupliquée d'un facteur de 1,5, à flexibilité comparable.

Nous avons choisi de rendre possible l'ajout d'opérateurs supplémentaires propres à chacune des voies ainsi que des registres. L'utilisation de ces ressources reste transparente pour le programmeur.

Accès aux données orthogonal Comme nous l'avons vu précédemment, l'utilisation d'un processeur orthogonal permet de réduire significativement le nombre d'instructions nécessaires à l'exécution d'un programme. Nous avons aussi vu que l'utilisation de données externes sur les deux opérandes d'une même opération binaire ne se produit que rarement sur l'ensemble de la chaîne de traitement. Aussi, avons nous défini que l'accès à la valeur d'un seul des deux opérandes de chaque voie doit être orthogonal, la valeur du second étant systématiquement imposée comme issue de la file de registres.

4.3.2 Définition du jeu d'instructions

Le jeu d'instructions (Instruction Set) est la partie du processeur qui est rendue visible au programmeur. Nous avons défini un jeu d'instructions suffisamment modulaire pour pouvoir l'adapter aux besoins des applications et aux contraintes architecturales. Pour cela, nous nous basons sur le choix des opérateurs permettant de couvrir les applications de traitement d'image comme nous l'avons fait précédemment. Toutes les instructions du processeur *SplitWay* s'exécutent en un seul cycle processeur.

Comme le processeur implémenté est un processeur VLIW deux voies, les instructions VLIW sont composées de deux « instructions simples » concaténées, chacune correspondant à une opération. Nous ne décrirons ici que le format des « instructions simples » qui est identique pour chacune des voies, puis nous édicterons les règles de leur utilisation sur deux voies.

4.3.2.1 Définition des instructions utilisateurs

• Les opérations Comme l'instruction sert avant tout à demander au processeur d'exécuter une opération, le champ principal désigne l'opération que nous nommerons `OP`. Il peut, par exemple, prendre les valeurs de la liste non exhaustive qui suit:

- `LD`: copie d'une donnée dans un registre;
- `ADD`: addition;
- `SUB`: et `CMP` soustraction;
- `MUL`: multiplication;
- `STR`: écriture dans le plan mémoire (adresse stockée dans un registre);
- `SHIFTR`: et `SHIFTL` décalage à droite et à gauche;
- `CMP`: comparaison;
- `JMP`: saut;
- `AND`: ET bit à bit;
- `OR`: OU bit à bit;
- `NOP`: attente d'un cycle processeur.

La plupart de ces instructions utilisent des opérandes (à l'exception de `NOP`) et génèrent un résultat. Pour Pour générer ce résultat, il est nécessaire d'avoir des données sur lequelles opérer, c'est pourquoi nous avons prévu différents arguments au mot d'instruction.

• Les arguments - `DEST` Le premier argument est le numéro du registre où sauver le résultat de l'opération. Il est codé dans le champ `DEST` dont la taille en bits est donc au moins le log_2 du nombre de registres. Dans le cas de l'opération `STR` qui sauvegarde une valeur en mémoire externe, ce champ permet de stocker le numéro du registre contenant l'adresse où écrire. Certaines opérations ne nécessitent pas la sauvegarde d'une valeur dans un registre, comme par exemple la comparaison `CMP`, dans ce cas, sa valeur n'est pas utilisée.

• Les arguments - `A` La plupart des instructions ont besoin d'un ou plusieurs arguments sur lesquels opérer. Le champ `A` permet de coder le numéro du registre dans lequel est stocké la valeur du premier opérande. Ce champ est de la même taille que le champ `DEST`, pour les même raisons liées au nombre de registres. Avec les champs que nous avons présentés, il est possible de traiter les opérations unaires (incrément, décrement etc.) Notons par ailleurs que `A` peut être le même registre que `D`, l'opération sera exécutée sur la valeur du registre de `A`, puis remplacée ensuite.

• Les arguments - `B` Afin de traiter des opérations binaires utilisant deux opérandes, il est nécessaire d'introduire un champ supplémentaire `B`. Comme le champ `DEST` et `A`, il code le numéro du registre du second opérande, avec les mêmes contraintes sur sa taille que celles du champ `A`. Toutefois, Avec ces champs, seules les opérations de registre à registre sont possibles. Or, nous avons vu précédemment qu'il est essentiel de pouvoir utiliser des données externes, comme celles disponibles dans les registres de voisinages.

• Accès aux données Aussi, avons-nous défini l'un des deux opérandes comme orthogonal, en l'occurrence B, c'est à dire que sa valeur peut être indifféremment issue d'une source externe ou de la file de registres. La méthode que nous avons conçue et employée ici reste extensible à A mais au prix d'une augmentation de la taille du mot d'instruction et de la surface silicium, rédhibitoire au regard de l'augmentation des performances apportées. Pour ce mode d'accès, un champ supplémentaire appelé A_Source est introduit ce qui permet de coder la source de la donnée. Ce champ peut, par exemple, prendre une des valeurs suivantes :

- R pour une donnée issue de la file de registres;
- C pour une donnée constante issue du mot d'instruction;
- V pour une donnée issue du gestionnaire de voisinages;
- P pour une donnée issue d'un processeur voisin;
- M pour une donnée issue du plan mémoire;
- B pour une donnée issue d'un résultat d'une instruction précédente;
- D pour une valeur d'un des drapeaux [1] permettant de gérer les conditions;
- e1, e2 à eN pour des données issues d'une des N entrées disponibles.

Ces champs supplémentaires permettent de sélectionner une donnée parmi les entrées du processeur *SplitWay*. Il est souvent nécessaire d'adresser une valeur de ces entrées. Par exemple, lorsque le programmeur choisit l'entrée correspondant au gestionnaire de voisinages en précisant V comme source de données, il lui est nécessaire de transmettre les coordonnées du voisinage à lire.

Prenons par exemple le cas où le gestionnaire de voisinages gère des voisinages 15×15. Le programmeur pourra souhaiter lire le pixel haut gauche (soit le pixel de coordonnées $(-7,-7)$). Ces valeurs sont transmises directement dans le mot d'instruction, ce qui lui permet d'avoir dans le même cycle toutes les informations nécessaires pour lire les données. Pour cela, plutôt que d'ajouter un champ supplémentaire, nous avons choisi d'étendre le champ B (aussi utilisé pour coder le numéro de registre de l'opérande B). Sa nouvelle taille dépend du nombre de données qui doivent être adressées.

Par exemple une taille de 7 bits autorise l'accès à des matrices de voisinages 11×11 tandis que 8 bits permettent d'accéder à des matrices de taille 16×16. Comme cet exemple d'accès aux voisins le montre, la taille du champ B impose la limite de l'espace externe qu'il est possible d'adresser en lecture. C'est pourquoi nous avons intégré la possibilité d'utiliser des valeurs de la file de registres en tant qu'adresses. Pour implémenter cette méthode, nous avons choisi de considérer que la partie basse du champ B contient le numéro d'un registre dans lequel est stockée l'adresse à considérer, si son bit le plus significatif est placé à 1. Pour ne pas intégrer un bit supplémentaire au champ B, il est possible de n'utiliser que la moitié de la file de registres pour adresser des valeurs externes. Par exemple, si, B à une taille de 5 bits, et que l'on souhaite signaler que la valeur du voisin à accéder est stockée dans le registre numéro 3, le champ B prendra les valeurs suivantes « 1 $\underbrace{0011}_{\text{Valeur du registre}}$ ».

De plus, un registre spécial accessible est utilisé comme adresse de base pour tout accès au plan mémoire. Sa valeur est systématiquement ajoutée à l'adresse demandée, qu'elle soit dans le champ B ou issue de la file de registres. De plus, nous avons ajouté un registre spécial dont la valeur est systématiquement concaténée à celle du champ B lorsqu'il s'agit d'adresser des valeurs. Cette méthode nous permet d'accéder à des espaces d'adresses dont la taille dépasse la dynamique des registres.

1. Les instructions conditionnelles s'appuient sur ces drapeaux, comme cela sera détaillé plus en avant.

Gestion des métadonnées Nous avons vu qu'il était possible d'extraire de manière transparente des métadonnées qui sont stockées au sein d'un mot de données. Pour cela, plusieurs registres de masques de données correspondant à toutes les métadonnées disponibles sont configurés par le programmeur au niveau de la tuile de calcul. Le programmmeur réalise ensuite les accès aux données à l'aide d'un de ces masques. Il lui suffit d'associer le numéro du masque à utiliser sur l'opérande. L'utilisation de ces métadonnées permet en outre de travailler sur des mots de données de taille plus importante que celle du chemin de données, puisqu'ils peuvent être décomposés en plusieurs champs qui peuvent être traités séparément. Le support des métadonnées permet au programmeur de définir une partie entière et une partie décimale, ou encore une partie réelle et une partie imaginaire pour ses données.

Trois champs supplémentaires sont introduits pour adresser ces masques (un par opérande), `A_fmt` pour la première opérande `A`, `B_fmt` pour la seconde opérande et `Res_fmt` pour le résultat `Res`. Il est envisageable d'utiliser le même masque pour les deux opérandes, et un pour le résultat, réduisant alors la taille du mot d'instruction. Au niveau du langage assembleur, le numéro du masque à utiliser est associé à la notation pointée (aucune valeur signifie que l'ensemble des bits est à exploiter). Par exemple, en tant qu'opérande, `R2.4` fait référence au registre `R2`, pour lequel la métadonnée est extraite grâce au masque `4`. S'il est utilisé en tant que registre, alors seuls les bits de cette métadonnée seront écrits en `R2` en conservant les autres parties du registres.

Instructions conditionnelles Nous avons maintenant la possibilité d'exécuter des instructions séquentiellement et d'utiliser des sauts. Toutefois, afin de traiter les instructions de type `IF/ELSE` ou encore les `CASE`, il est nécessaire de supporter des instructions conditionnelles. C'est pourquoi nous avons introduit la notion de drapeau ainsi que des champs pour coder des conditions.

Les drapeaux sont constitués de *NbDrapeaux* bits que le programmeur peut positionner sur le résultat d'une instruction de son choix. Il peut ensuite utiliser leur valeur pour conditionner l'exécution de ses instructions. Pour cela, il est prévu de permettre le positionnement à `VRAI` des drapeaux dans les cas suivants :

- `MZ` si le résultat précédent est un zéro;
- `MNZ` si le résultat précédent n'est pas un zéro;
- `MS` si le résultat précédent est supérieur à zéro;
- `MNS` si le résultat précédent est inférieur à zéro;
- `MC` si le résultat précédent génère une retenue;
- `MNC` si le résultat précédent ne génère pas de retenue;
- `MO` s'il y a dépassement de capacité sur le résultat précédent;
- `MNO` s'il n'y a pas dépassement de capacité sur le résultat précédent.

Le champ appelé `CxPosDrap` permet de coder les choix de position des drapeaux, tandis qu'un champ `NumDrap` permet de coder le numéro du drapeau à mettre à jour. Sa taille en bits est d'au moins de $log_2(NbDrapeaux)$. Ils conservent leurs valeurs tant qu'ils ne sont pas explicitement repositionnés par le programme. Par ailleurs le champ `CxPosDrap` permet de coder les choix de position des drapeaux. Dans notre cas, sa taille est de 3 bits, puisqu'il sert à coder les huit choix listés précédemment.

Nous avons maintenant permis au programme de positionner des drapeaux à `VRAI` sur le résultat d'une opération. Afin de les exploiter pour rendre l'exécution des instructions conditionnelles, deux champs supplémen-

taires sont ajoutés pour rendre conditionnelles les opérations. Le premier `NumDrapEx` est le numéro du drapeau qui la conditionne. Le second `CondEx` code la condition à exploiter, c'est à dire :

- `VRAI` exécute l'opération si le drapeau est positionné à `VRAI`;
- `FAUX` exécute l'opération si le drapeau est positionné à `FAUX`;
- Aucune précision permet d'exécuter l'opération quelle que soit la valeur du drapeau.

Dès lors, il est possible d'exécuter des instructions conditionnelles, y compris lorsque plusieurs niveaux d'imbrications sont en jeu.

Jeu d'instructions VLIW 2 voies Le processeur *SplitWay* est un processeur VLIW deux voies, ce qui signifie que deux instructions telles que celles décrites précédemment peuvent être exécutées en parallèle. Comme l'ensemble des ressources sont indifféremment accessibles entre les deux voies (la file de registres, les drapeaux, les opérateurs, etc.), il est nécessaire de définir des règles de programmation permettant d'assurer leur bon fonctionnement. Ainsi, pour deux instructions destinées chacune à une voie, *i* et *j* qui s'exécutent simultanément, il est nécessaire de veiller au respect des règles édictées ci dessous en 4.9 :

$$\texttt{DEST}_i \quad \texttt{Op}_i \quad \ldots \quad \texttt{NumDrap}_i \;\|\; \texttt{DEST}_j \quad \texttt{Op}_j \quad \ldots \quad \texttt{NumDrap}_j \tag{4.9}$$

1. écrire dans le même registre est interdit donc $\texttt{DEST}_i \neq \texttt{DEST}_j$;
2. assigner le même drapeau sur les deux voies est interdit donc $\texttt{NumDrap}_i \neq \texttt{NumDrap}_j$;
3. l'utilisation de la même instance d'un même opérateur simultanément est interdit.

Si malgré les outils logiciels ces règles ne sont pas respectées, la première voie (la voie *i*) du processeur VLIW est prioritaire sur la seconde (la voie *j*). Par ailleurs, lorsque les instructions conditionnelles sont utilisées, l'écriture de deux instructions utilisant les mêmes ressources est permise comme par exemple le traitement d'un `IF/ELSE` durant le même cycle processeur.

Exemple d'utilisation du jeu d'instructions Le premier exemple consiste à binariser une image, et à transmettre l'information en tant que métadonnée au pixel. Le format de la donnée d'entrée est un mot sur lequel le pixel est codé sur les 8 bits de poids faible. Le mot de donnée de sortie est un mot composite, où les 8 premiers bits donnent la valeur du pixel et un bit supplémentaire de métadonnée contient la valeur seuillée de l'image. La version séquentielle et la version *SplitWay* de cet algorithme simple sont présentés en Figure 4.8.

Cet algorithme peut être décrit sur les deux voies comme le montre la seconde partie de cette même figure. Si la condition (comparaison du seuil) est vraie, le drapeau numéro « 0 » est positionné à `VRAI`. Alors la première voie traite l'opération grâce au mot clé `VRAI F0` qui vérifie l'état du drapeau « 0 ». La seconde voie est inhibée, puisque la condition pour qu'elle s'exécute est que le drapeau soit positionné à `FAUX`. Dans le cas où la condition est fausse, alors la condition inverse est exécutée. Il est donc possible de traiter dans le même cycle une instruction `IF/ELSE`.

Dans cet exemple, l'utilisation des métadonnées supprime toute opération logique, ainsi ce traitement qui nécessiterait cinq à six instructions et autant de registres, est décrit en trois cycles, dont un dédié à l'initialisation d'une constante, et un seul registre est utilisé.

```
1: seuil=128
2: if Pixel(0,0) = seuil then
3:     Bin = 0
4: else
5:     Bin = 1
6: end if
7: Bin = Bin « 9
8: Sortie = Pixel OR Bin
```

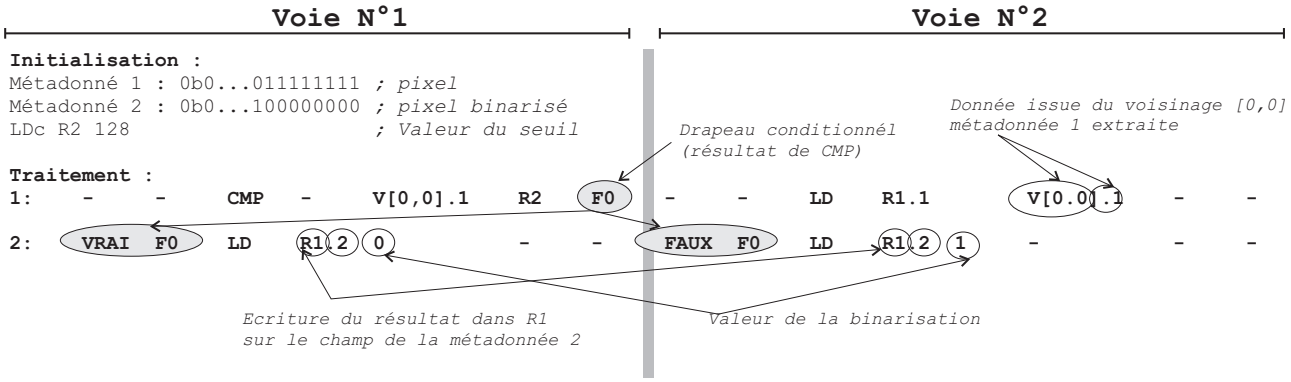


FIG. 4.8: EXEMPLE D'UN ALGORITHME DE BINARISATION, LA PARTIE IF EST RÉALISÉE SUR LA PREMIÈRE VOIE, TANDIS QUE LA PARTIE ELSE EST RÉALISÉE SUR LA DEUXIÈME VOIE. LA MÉTADONNÉE 2 PERMET D'ENREGISTRER LA VALEUR DU BIT CONJOINTEMENT À LA VALEUR DU PIXEL.

4.3.3 Accès aux données

Nous avons présenté précédemment le choix des opérateurs, puis le jeu d'instructions tel qu'il est utilisable par le programmeur. Le mode d'accès aux données orthogonal est construit autour d'un multiplexeur sur lequel sont connectés les signaux d'entrées.

Mode d'accès orthogonal en lecture Ce multiplexeur est commandé par le champ SOURCE de l'instruction. C'est ainsi que la donnée issue de la source définie dans l'instruction est sélectionnée, tandis que l'adresse de la valeur à lire est transmise à partir du champ B. Par exemple, lorsqu'il faut lire une donnée issue du gestionnaire de voisinages, le code correspondant aux coordonnées du voisinage est transmis au gestionnaire de voisinages. Ce même principe s'applique à toutes les sources de données du processeur, comme la file de registres, ou encore le plan mémoire sur lequel peuvent être connectés une mémoire de travail ou des coprocesseurs.

Accès en écriture Comme cette technique permet les accès en lecture seule, il est nécessaire d'autoriser des accès en écriture vers des composants externes, et notamment la mémoire de travail. Pour cela, nous avons ajouté un opérateur de stockage auquel est associée l'opération STR. Celui-ci a un accès direct au plan mémoire et donc aux composants qui lui sont connectés.

Espace mémoire adressable La taille du plan mémoire est déterminée par la largeur du chemin de données défini à l'instanciation du processeur *SplitWay*. De plus, un registre spécial permet de définir l'adresse de base dans laquelle tous les accès en lecture comme en écriture sont définis. Sa taille est au plus celle de la largeur du chemin de données du processeur *SplitWay*. Si sa largeur est de L bits, alors, $2^{2\times L}$ mots de L bits sont au maximum adressables. Il est ainsi possible de connecter, grâce à ce plan mémoire, une mémoire de travail, des LUTs, mais aussi différents composants externes comme des accélérateurs, un bus etc.

Autres modes d'accès Par ailleurs, une partie de la file de registres est rendue accessible en lecture par des composants externes au processeur. Cette méthode est utilisée pour la communication de processeur à processeur, ou encore pour générer le flux de sortie de la tuile de calcul à partir des valeurs des résultats de calculs.

4.3.4 Réalisation du Pipeline

Comme nous l'avons vu, la consommation électrique des circuits en technologie CMOS diminue avec la fréquence, c'est pourquoi nous considérons des faibles fréquences de fonctionnement. L'état de l'art nous a permis de voir que les architectures les plus efficaces en consommation sont généralement cadencées à moins de 300 MHz. La fréquence maximale de fonctionnement d'un processeur est donnée par le chemin critique. Les processeurs sont conçus avec d'autant plus d'étages de pipeline qu'il faut maintenir un chemin critique faible, et permettre ainsi d'atteindre des fréquences de fonctionnement élevées. Dans notre cas, la pression sur le chemin critique est faible puisque nous ne cherchons pas à obtenir des fréquences de fonctionnement élevées. C'est pourquoi nous proposons de limiter le nombre d'étages du pipeline du processeur *SplitWay* à trois. Ce sont les trois étages FETCH, DECODE et EXECUTE, visibles en Figure 4.9. Non seulement l'ajout d'étages supplémentaires serait inutile au regard de la fréquence de fonctionnement visée, mais il induirait en plus un surcoût en surface silicium (barrières de registres *bypass*) etc. Notons que si la fréquence de fonctionnement doit être significativement augmentée, il suffit de diviser ces trois étages afin de réduire le chemin critique. Le processeur possède différents modes de gestion de la consommation au niveau architectural qui sont décrits en dernière partie.

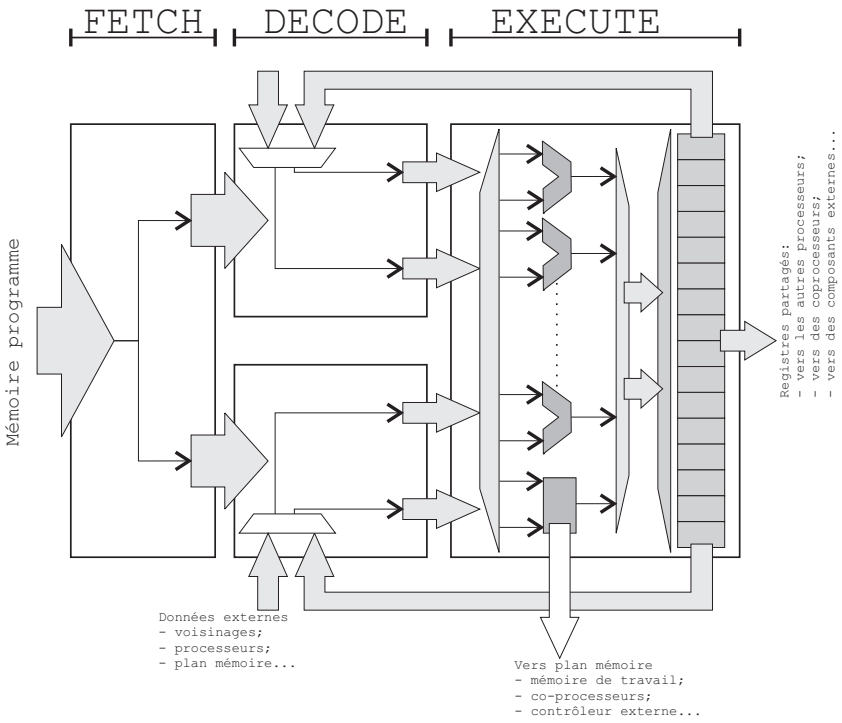


FIG. 4.9: SCHÉMA SYNOPTIQUE DU PROCESSEUR *SplitWay*.

4.3.4.1 L'étage FETCH

L'étage FETCH a pour rôle principal de lire les mots d'instructions depuis la mémoire programme pour les transmettre à l'étage suivant du pipeline. C'est à ce niveau que les sauts sont gérés et notamment les sauts

conditionnels exécutés en fonction des résultats de précédentes instructions. Il possède à cette fin un registre spécifique contenant les différents drapeaux qui sont commutés sur les résultats de précédentes instructions. Le programmeur choisit explicitement de commuter tel ou tel drapeau sur le résultat de l'opération de son choix. Il permet également de démarrer l'exécution de différents segments de code donnés en fonction des événements extérieurs, comme par exemple l'arrivée d'un nouveau pixel.

4.3.4.2 L'étage DECODE

L'étage DECODE est constitué de deux unités identiques permettant le décodage des instructions (une pour chaque voie). Chacune des voies reçoit la partie du mot d'instruction VLIW qui lui est attribué. Ce dernier est décomposé et les différents champs de l'instruction sont extraits. Les opérandes sont alors lus afin de présenter à l'étage suivant les valeurs numériques prêtes à être calculées. Le décodeur évalue pour chaque opérande, s'il provient de la file de registres ou du résultat d'une des deux précédentes opérations ce signifie qu'il ne sera pas écrit en file de registres avant le cycle suivant, un retour direct de ces valeurs vers le décodeur est donc prévu. La gestion de la lecture des métadonnées et le mode d'accès orthogonal sont aussi implémentés à ce niveau.

Lorsqu'une instruction conditionnelle se présente, elle est évaluée par une comparaison du bit d'un des drapeaux que le programmeur aura précédemment assigné par rapport au résultat d'une opération précédente. Si la condition est vraie, alors l'instruction est transmise à l'étage EXECUTE, si elle est fausse, elle est remplacée par une instruction vide.

4.3.4.3 L'étage EXECUTE

L'étage EXECUTE contient l'ensemble des opérateurs du processeur. Comme nous l'avons vu ci-dessus, ceux-ci sont mutualisés pour les deux voies. Cette technique ne permet pas d'utiliser simultanément le même opérateur sur les deux voies, il est donc nécessaire d'intégrer deux instances du même opérateur pour pouvoir exécuter deux opérations du même type en parallèle.

Les deux résultats d'opérations, un pour chaque voie, sont directement écrits dans les registres de destination au cycle processeur suivant. Contrairement à de nombreuses architectures, le processeur *SplitWay* ne possède pas d'accumulateur en sortie des opérateurs, les résultats sont donc directement enregistrés dans la file de registres durant le cycle suivant leur exécution. Ces résultats sont utilisés pour mettre à jour les drapeaux qui servent à déterminer si une instruction conditionnelle doit être exécutée ou non. Cette mise à jour doit être spécifiée par le programmeur. Leur nombre est déterminé à la création du processeur ; typiquement 4 à 8 drapeaux permettent une programmation souple et efficace en terme de nombre d'instructions des algorithmes de traitement d'image que nous avons étudiés.

4.3.5 Gestion de la consommation

Outre le contrôle de la fréquence de fonctionnement du processeur, plusieurs modes ont été mis en œuvre au niveau du processeur *SplitWay* pour limiter sa consommation électrique. Ceci est réalisé de deux manières, la première en inhibant l'horloge des étages suivants et la seconde en limitant les transitions de leurs transistors que nous sollicitons.

4.3.5.1 Le mode « sommeil profond »

Le mode « sommeil profond » consiste à inhiber l'horloge de tous les éléments du processeur et à désactiver les interfaces de communication. Un délai minimum de cinq cycles processeurs est nécessaire pour permettre

la réactivation du processeur. Ces cinq cycles sont utilisés pour réinitialiser la configuration minimale du processeur, si elle a changé durant son sommeil profond. Ce mode n'est utilisé que lorsque plusieurs dizaines de cycles processeurs sont inutilisés. De plus, le réveil nécessite une intervention extérieure qui peut introduire une latence supplémentaire, ou lancer une reconfiguration complète (effacement des mémoires, réinitialisation des LUTs etc.) si cela est nécessaire.

4.3.5.2 Le mode « sommeil »

Comme le redémarrage du processeur en sortie du mode « sommeil profond » nécessite plusieurs cycles et une intervention extérieure, le mode « sommeil » a été défini. Ce mode est basé sur le même principe que le précédent à la différence près qu'il est commandé par l'étage FETCH. De cette manière, le processeur gère de manière autonome son sommeil et son réveil.

À chaque fin de traitement, l'horloge des étages DECODE, EXECUTE, et de la file de registres sont inhibées par l'étage FETCH. Le redémarrage nécessite un cycle processeur, le processeur est donc prêt à continuer l'exécution d'un programme, ou bien à vider son pipeline. Ce mode est typiquement utilisé lorsque le traitement d'un pixel est terminé et qu'il faut attendre l'arrivée du pixel à traiter suivant.

4.3.5.3 Le mode « léthargie » ou « attente »

Le mode « léthargie », appelé aussi « attente », consiste quant à lui à maintenir le processeur en activité, en conservant tous les signaux à leurs valeurs courantes. Le nombre de commutation des transistors est ainsi minimisé. Seuls les signaux qui induisent une modification de l'activité du processeur sont modifiés: aucune opération n'est réalisée, et aucune donnée n'est lue. Ainsi, l'horloge du processeur reste le seul signal actif.

Ce mode peut être activé de l'extérieur, ou automatiquement par chaque étage du pipeline dès qu'aucune action ou opération n'est à réaliser. Il est utilisé dès que possible, comme par exemple lors de l'utilisation d'instructions conditionnelles qui n'amène pas à l'exécution d'une opération.

4.3.6 La file de registres

La file de registre est composée de plusieurs registres de même taille que celle du chemin de données. Elle possède quatre ports de lecture, chacun correspondant aux deux opérandes lus par les deux voies DECODE. De plus, la file de registres possède deux voies en écriture permettant aux résultats des deux opérations s'exécutant simultanément d'être écrits en parallèle dans deux registres de destination différents.

Par ailleurs, certains registres sont utilisés pour transmettre des valeurs à des composants extérieurs, comme par exemple les processeurs voisins. Afin de limiter le coût en surface silicium de ces accès, seuls des accès en lecture sont rendus possibles sur ces registres. Par exemple, lorsque le processeur *SplitWay* est utilisé dans une structure SIMD, un registre est connecté avec les processeurs « Est » et « Ouest » et trois registres généraux sont utilisés comme registres de sortie en étant connectés avec l'unité d'entrée-sortie qui permet la reconstruction d'un flux.

4.4 Support des métadonnées

Nous avons vu que la possibilité d'intégrer et de traiter des données composites apporte une souplesse appréciable en traitement d'image. En effet, les données à traiter sont généralement des composantes couleur pour

un même pixel ou des métadonnées qui sont ajoutées à la valeur du pixel. Comme les formats de composition des données à manipuler peuvent varier d'un traitement à l'autre, et que ce dernier n'est pas normalisé, nous avons mis en place un méthode pour permettre au programmeur un accès transparent à ces données.

La première partie de cette section rappelle le principe du mécanisme mis en place ainsi que la technique utilisée pour rendre les formats de données paramétrables. La seconde partie présente l'implémentation matérielle du module permettant de réaliser les accès en lecture et en écriture aux valeurs de ces données.

4.4.1 Mécanisme d'accès aux métadonnées

Chaque tuile de calcul peut recevoir des flux de données qui sont composés de différentes manières. Par exemple une donnée peut intégrer plusieurs composantes (rouge, vert et bleu, luminance et chrominance) d'un même pixel, voire de plusieurs pixels produits par différents traitements ou issus d'un système de stéréo-vision.

4.4.1.1 Exemple d'utilisation des métadonnées

Certains algorithmes sont réalisés en plusieurs étapes pour lesquelles il est nécessaire de calculer des résultats intermédiaires que les étapes suivantes utilisent pour réaliser le traitement – l'orientation du gradient d'un pixel, la direction du flot optique etc. D'autres algorithmes nécessitent le passage dans l'espace de Fourier, les données manipulées sont alors des nombres complexes. De la même manière, un programmeur peut souhaiter manipuler séparément des parties décimales et des parties entières. Ces métadonnées peuvent être utilisées pour écrire la position dans un autre espace de coordonnées du pixel manipulé. Enfin il peut être utile d'associer des métadonnées aux valeurs des pixels pour les niveaux applicatifs supérieurs comme par exemple des points d'intérêt, ou encore la notion de profondeur lorsqu'il s'agit d'un système de stéréo-vision.

4.4.1.2 Accès aux métadonnées

Traditionnellement, l'accès aux données composites est réalisé à l'aide de masques et d'opérations binaires par le programmeur, tandis que certaines architectures fixent le type et le format des données dès la conception du système. Réaliser ces accès aux données de manière logicielle implique un surcoût pouvant atteindre plusieurs opérations par pixel. Or nous avons vu qu'il était souhaitable de minimiser le nombre d'opérations à exécuter par pixel afin de réduire la fréquence de fonctionnement et donc la consommation électrique. Les solutions dédiées quant à elles ne sont pas envisagées en raison du large éventail de formats possibles.

C'est pourquoi nous avons choisi de rendre paramétrable les différents formats de données que le programmeur est susceptible d'utiliser au sein d'une même tuile de calcul. Il définit donc lors de l'initialisation de la tuile de calcul les formats des différents champs de données dans des registres de masque. Ensuite, la manipulation des données est réalisée en spécifiant le numéro du registre de format à utiliser, comme nous l'avons vu lors de la description du jeu d'instructions. L'accès aux métadonnées, que ce soit en lecture ou en écriture n'introduit pas de surcoût algorithmique. L'intégration de cette technique aux tuiles de calcul rend possible des constructions architecturales complexes sans que cela n'impacte négativement l'écriture des programmes.

4.4.1.3 Implications architecturales

Le mode SIMD est mal adapté à l'exécution des prédicats de type `CASE`, c'est pourquoi nous utilisons le mode Multi-SIMD. De tels prédicats sont généralement appelés sur les résultats d'une précédente opération. Nous avons donc permis de contrôler le choix du segment de code à exécuter pour un pixel donné par la valeur d'une métadonnée calculée dans une tuile précédente. Ainsi, le programmeur n'a pas à exécuter d'opérations

conditionnelles pour l'implémentation des algorithmes basés sur des prédicats de type `CASE`, mais seulement à diviser son programme en plusieurs segments comme cela a été expliqué dans le chapitre 3. Dans certaines circonstances, la largeur du mot de données entrant dans la tuile de calcul peut être supérieure à la taille du chemin de données du processeur *SplitWay*. Par exemple, un mot composé de trois composantes de 16 bits chacune représente 48 bits. L'accès direct aux valeurs des composantes permet de ne pas instancier une version 48 bits du processeur de calcul *SplitWay*, ou d'autres ressources de la tuile de calcul. L'utilisation des métadonnées permet aussi d'envisager la gestion des dépassements de capacité en scindant les mots de données de taille importante en deux, ou bien de permettre l'utilisation de mots de registres d'une taille différente de celle du chemin de données.

D'une manière générale, la taille des différentes données manipulées par la tuile de calcul peuvent être différentes. Il est ainsi envisageable de proposer une tuile de calcul intégrant des processeurs dont la taille du chemin de données est de 16 bits, associée à des registres 32 bits, connectée à une mémoire de travail de mots de 64 bits et dont le flux de données est composé de pixels codés sur 24 bits. Le programmeur devra simplement préciser le format des données pour des mots pouvant atteindre jusqu'à 64 bits. Ceci est possible grâce aux registres de masques.

4.4.2 Implémentation de l'accès aux métadonnées

L'implémentation de l'accès aux métadonnées passe d'abord par la définition des registres de masques. Ces derniers sont d'abord introduits, puis le mode d'accès en lecture est ensuite présenté, suivi du mode d'accès en écriture.

4.4.2.1 Présentation des registres de masques

Chaque tuile de calcul intègre plusieurs registres de masques. Ces registres de taille identique à celle des mots de donnée, permettent au programmeur de prédéfinir les formats potentiellement utilisables au sein du programme qui est porté sur la tuile de calcul. Il peut être envisagé de rendre leur valeur modifiable durant l'exécution d'un traitement (ce qui peut accélrer une émulation de calcul en virgule flotante par exemple), bien que cela n'ait pas été implémenté. Seul le registre de format qui contrôle le choix des segments de code à exécuter en mode Multi-SIMD ne peut être modifié. L'implémentation de 4 à 8 registres de format permet de couvrir la majorité des cas d'utilisation.

Ces registres de format contiennent d'une part le masque d'accès aux valeurs des métadonnées, d'autre part, la position de la métadonnée qui est utilisée pour recaler la donnée afin qu'elle puisse être utilisée pour réaliser des calculs. Comme on peut le voir sur la Figure 4.10 qui présente quelques exemples de registres de formats, le masque est décrit dans une partie première du registre de masque, et sa position dans le mot de données dans une seconde partie, ce qui permet de décaler les données en conséquence lors de leurs accès.

4.4.2.2 Méthode d'accès aux métadonnées en lecture

L'accès aux données en lecture est réalisé directement dans le chemin de données. La méthode qui est décrite en Figure 4.11a consiste à masquer la donnée contenant la métadonnée à extraire avec les valeurs du masque `Mi`. Il en résulte la valeur de la métadonnée, toutefois, celle-ci n'est pas directement exploitable par les opérateurs. En effet, les champs qui n'ont pas été sélectionnés sont remplacés par des `0`. C'est pourquoi la valeur est alignée en propageant l'éventuelle retenue puisque nous traitons des entiers signés. Deux opérations sont donc introduites sur le chemin de données.

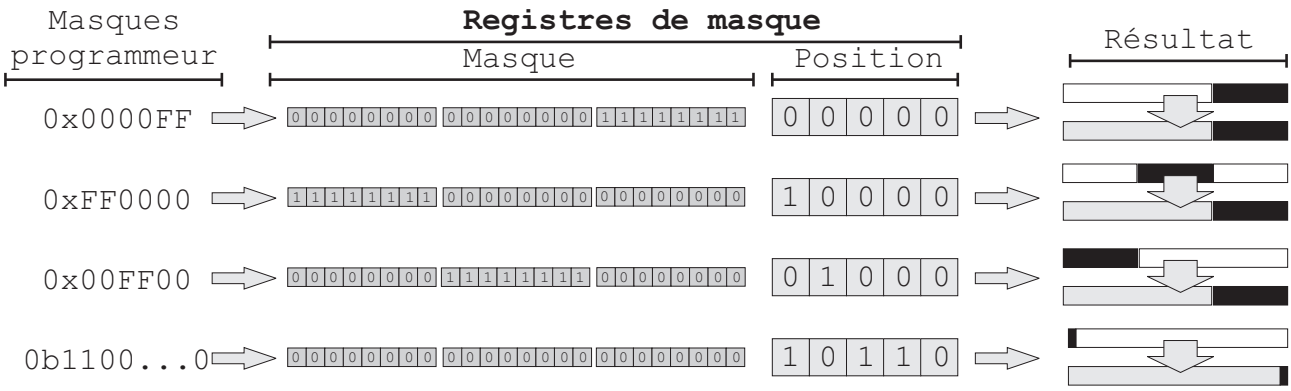
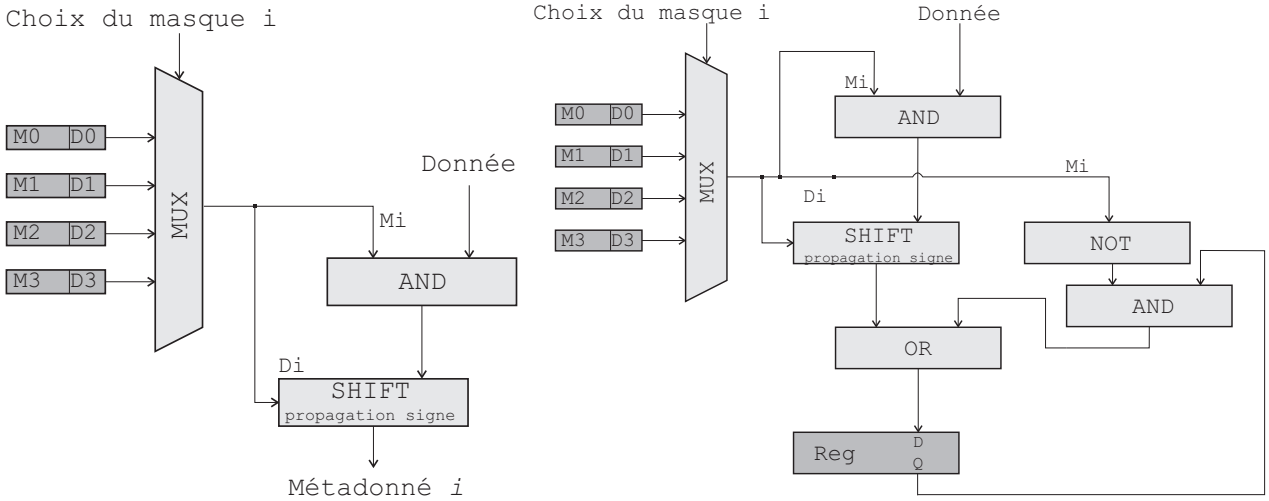


FIG. 4.10: EXEMPLES DE QUELQUES REGISTRES DE MASQUES UTILISÉS POUR LES MÉTADONNÉES. DE GAUCHE À DROITE, LES MASQUES TELS QU'ILS SONT DÉCRITS PAR LE PROGRAMMEUR, LES MASQUES TELS QU'ILS SONT IMPLÉMENTÉS, ET LE RÉSULTAT DE L'EXTRACTION D'UNE MÉTADONNÉE AVEC CHACUN DE CES MASQUES.

Pour un maximum de flexibilité, l'accès en lecture est intégré à l'entrée de chaque opérande au niveau du processeur *SplitWay*. De plus, un module de lecture supplémentaire doit être intégré entre l'unité de contrôle et chaque processeur afin qu'il puisse sélectionner les instructions qui lui sont destinées à partir de la valeur d'un champ de métadonnées.

4.4.2.3 Méthode d'accès aux métadonnées en écriture

L'écriture d'une valeur d'une métadonnée nécessite plus d'opérations que pour sa lecture. En effet, il est essentiel de ne pas écraser les valeurs des autres champs de données que contient le registre dans lequel l'écriture doit être réalisée. Dans un premier temps les valeurs des autres métadonnées sont extraites. Celle de la métadonnée à traiter est ensuite insérée avant d'être écrite dans le registre. La Figure 4.11b présente cette technique, qui engendre un surcoût en terme de surface silicium au niveau du processeur. Un module pour chaque voie de



(a) ACCÈS EN LECTURE À UNE MÉTADONNÉE i. (b) ACCÈS EN ÉCRITURE À UNE MÉTADONNÉE i.

FIG. 4.11: LOGIQUE PERMETTANT L'ACCÈS AUX MÉTADONNÉES i ASSOCIÉES PAR LES REGISTRES DE MASQUES Mi DANS LES CAS (a) D'ACCÈS EN LECTURE ET (b) D'ACCÈS EN ÉCRITURE.LES NOMBRES SIGNÉS SONT SUPPORTÉS EN PROPAGEANT LA VALEUR DE BIT DU SIGNE LORS DU DÉCALAGE DES VALEURS.

calcul est destiné à être intégré à la sortie du module d'exécution. L'utilisation de ce module engendre des délais d'écriture importants. Pour limiter cela, il peut être envisagé d'intégrer un étage de pipeline supplémentaire, mais cette solution augmente inévitablement la surface silicium du processeur.

4.4.2.4 Gestion des métadonnées simplifiée

Devant le surcoût architectural lié à une gestion complète des métadonnées, une version simplifiée est implémentée. Cette version permet le contrôle du mode Multi-SIMD de la tuile, mais aussi l'extraction des différents champs de données. Dans cette version, l'accès en lecture n'est possible que sur les données issues du gestionnaire de voisinages. Elle est réalisée par l'intégration de deux modules de lecture par processeur *SplitWay* sans que cela n'affecte son chemin critique. De plus un champ supplémentaire fixé à la création de la tuile de calcul permet le contrôle du mode Multi-SIMD sur des résultats de calculs précédents.

Ce flux est alors construit par concaténation itérative de différentes valeurs de registres utilisés. Ainsi, un seul module d'écriture simplifié de métadonnées par processeur (ou groupe de processeur) peut être utilisé en temps masqué (pendant que le traitement du pixel suivant est exécuté). Par exemple, si le flux de sortie doit être formé de trois composantes stockées dans respectivement trois registres, le programmeur a juste à définir les registres de format leur correspondant. Ainsi la valeur du premier registre est masquée, décalée, puis concaténée à la valeur du second registre et ainsi de suite. Cette opération est réalisée pendant que le calcul du pixel suivant est exécuté. Un registre spécifique de quelques bits est utilisé pour la partie permettant de définir le champ fixé pour le contrôle Multi-SIMD de la tuile de calcul suivante.

Conclusion sur la gestion des métadonnées

Cette section a présenté une technique pour accéder de manière transparente à différents champs de données. Les possibilités offertes par l'utilisation systématique d'une telle extension sont considérables. Toutefois, sa complexité architecturale la rend difficilement généralisable dans le cas d'un système fortement contraint. C'est pourquoi nous avons proposé une version simplifiée qui permet l'utilisation des métadonnées au niveau d'une tuile de calcul sans surcharger le processeur *SplitWay*. Ainsi le mode Multi-SIMD peut être contrôlé par un champ de quelques bits fixé à la création de la tuile de calcul.

4.5 Unité de contrôle

Une tuile de calcul est composée d'un groupe de processeurs *SplitWay* qui fonctionnent en mode SIMD ou Multi-SIMD. Par ailleurs, le gestionnaire de voisinages met en forme les données à partir du flux de pixels entrant et le présente aux processeurs sous forme de matrices intégrant le pixel à traiter et son voisinage. Dès que les données sont prêtes à être traitées, le gestionnaire de voisinages le signale à l'unité de contrôle qui transmet les instructions aux processeurs.

L'unité de contrôle d'une tuile de calcul transmet simultanément à tous les processeurs les instructions de traitement de pixels à partir des mémoires programme qu'elle intègre. Le mode de fonctionnement SIMD est utilisé pour exécuter simultanément les mêmes instructions sur l'ensemble des données tandis que le mode de fonctionnement Multi-SIMD permet d'exécuter des segments de code différents en fonction des pixels à traiter.

4.5.1 Modes de fonctionnement

C'est, par définition, le mode de fonctionnement SIMD, qui est utilisé pour exécuter simultanément les même instructions sur un ensemble de donées différentes. Toutefois, lorsque les données sont de nature dif-

férentes, par exemple lorsqu'il s'agit de traiter une image brute où sont alternées les couleurs des pixels sur un même flux, les codes à exécuter sur chacun d'entre eux diffèrent. C'est pourquoi le mode Multi-SIMD est utilisé, dans ce cas, l'unité de contrôle peut sélectionner le segment de code en fonction du type de données que chaque processeur doit traiter. Associer le traitement d'un seul pixel à un processeur *SplitWay*, et non pas un bloc de pixels, permet d'utiliser efficacement le mode Multi-SIMD. Comme chaque processeur de la tuile manipule un pixel, il suffit de veiller à lui transmettre les instructions correspondantes.

4.5.1.1 Mode de fonctionnement SIMD

Dans le cas SIMD, le même programme est transmis à tous les processeurs. Par conséquent, une seule unité de contrôle est utilisée. Elle intègre une mémoire programme et une unité FETCH qui transmet les instructions simultanément à l'ensemble des processeurs *SplitWay* d'une même tuile de calcul.

L'ensemble des processeurs de la tuile de calcul mutualisent donc le premier étage de pipeline qu'est l'unité FETCH, ainsi que la mémoire programme qui est associée. De fait, chaque processeur ne peut réaliser individuellement des instructions de saut. Celles-ci doivent être réalisées pour l'ensemble des processeurs d'une même tuile. Dans ce cas, l'exécution ou la non exécution de chaque opération doit être évaluée individuellement au niveau de chaque processeur.

4.5.1.2 Mode de fonctionnement Multi-SIMD

Dans le cas Multi-SIMD, différents segments de code sont transmis à différents groupes de processeurs d'une même tuile. Comme nous l'avons vu dans le chapitre 3, ce mode de fonctionnement est particulièrement adapté au traitement d'images brutes qui alternent des pixels de couleurs différentes. Plus généralement, ce mode de fonctionnement est parfaitement adapté à l'exécution des structures de type `CASE`.

Un couplage avec le gestionnaire de voisinages transmet à l'unité de contrôle les informations lui permettant de déterminer quel traitement réaliser sur chaque pixel. Dans ce cas, les mémoires programme doivent être lues indépendamment les unes des autres, ce qui correspond à l'utilisation de plusieurs étages FETCH. L'unité de contrôle doit donc connaître pour chaque pixel, et donc pour chaque processeur, quel segment de code associer.

4.5.2 Implémentation de l'unité de contrôle

Comme le mode SIMD n'est qu'un cas particulier du mode de fonctionnement Multi-SIMD, les unités de contrôle de chaque tuile de calcul sont implémentées avec les fonctions Multi-SIMD. Afin de limiter l'utilisation de la surface silicium et la consommation électrique, deux implémentations ont été réalisées, une première complète permettant un niveau de flexibilité élevé, et la seconde simplifiée se limitant au traitement d'images brutes et réduisant la surface silicium.

4.5.2.1 Caractéristiques générales

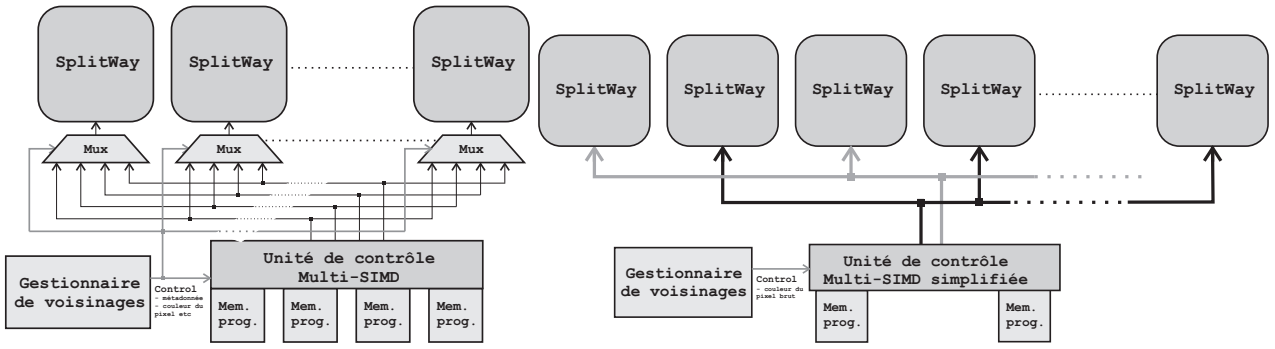
Le gestionnaire de voisinages reçoit les pixels et les organise pour que les processeurs puissent accéder à leurs valeurs. De la même manière, il a connaissance des délais d'intertrames, de fin de ligne etc. C'est donc le seul module de la tuile de calcul qui est à même de déterminer quand démarrer un nouveau traitement, et la nature de ce dernier.

En régime nominal, un traitement de pixel est démarré tous les *NbProcs* pixels, ce signal est transmis à l'unité de contrôle pour qu'il puisse démarrer un nouveau traitement. Ceci se traduit par le positionnement du (des) compteur(s) programme en fonction du traitement à réaliser. Par exemple les compteurs programme sont positionnés à l'adresse de début des segments correspondant aux différents cas d'exécution, ou bien, le cas échéant à celles des fins de trames ou de lignes.

4.5.2.2 Implémentation complète

En principe, chaque unité de contrôle doit intégrer autant de mémoires programme qu'il y a de cas `CASE` à potentiellement couvrir. Dans la pratique, 4 mémoires programme (ou deux mémoires double port) permettent de couvrir la majorité des cas d'utilisation. Elles sont toutes couplées au gestionnaire de voisinages qui leur transmet le numéro du segment de code à exécuter pour chaque pixel, et donc pour chaque processeur.

L'information, permettant de sélectionner le numéro de segment, peut être liée à la position du pixel, comme c'est le cas pour les images brutes, mais elle peut être issue d'un calcul précédent, et ainsi codée dans un champ de métadonnée spécifique. Le programmeur peut donc choisir entre le contrôle du mode Multi-SIMD sur la valeur d'un champ de métadonnée qu'il aura préalablement configuré, ou bien sur la position des pixels dans la matrice de couleur des images brutes (potentiellement tous les types de matrices sont supportés). La valeur permettant de commander le choix des segments de code vient contrôler le réseau de distribution des instructions aux processeurs comme on peut le voir en Figure 4.12a. Il peut aussi choisir de configurer l'unité de contrôle en mode SIMD. Dans ce cas, une seule unité FETCH est utilisée, et l'espace des mémoires programme est utilisable dans son ensemble pour permettre l'exécution de programmes plus conséquents.



(a) Mode Multi-SIMD complet.

(b) Mode Multi-SIMD simplifié.

FIG. 4.12: CONNEXION DE L'UNITÉ DE CONTRÔLE MULTI-SIMD AVEC LES PROCESSEURS *SplitWay* DANS LES CAS OÙ (a) LE MODE MULTI-SIMD COMPLET COMMANDÉ PAR LES MÉTADONNÉES OU LE TYPE DE PIXEL BRUT TRAITÉ, (b) LE MULTI-SIMD SIMPLIFIÉ EST ADAPTÉ AU TRAITEMENT D'IMAGE BRUTES.

4.5.2.3 Implémentation simplifiée

Lorsque les tuiles sont destinées à traiter des images brutes, le contrôle sur les métadonnées peut être supprimé. De même, le couplage avec le gestionnaire de voisinages peut être simplifié, puisque seule la position du premier pixel peut être transmis. La régularité des filtres de couleur utilisés permet de déterminer les couleurs des autres pixels, généralement par alternance. De même, le réseau de distribution des instructions est figé en alternant un processeur sur deux comme on peut le voir en Figure 4.12b. Les pixels traités étant contigus, seuls deux couleurs différentes se présentent simultanément, ainsi seules deux mémoires programme sont utilisées.

Conclusion sur le processeur de calcul

Cette section a décrit comment le mode Multi-SIMD était implémenté au sein de l'architecture *eISP*. Ce mode de fonctionnement permet de réduire notablement les instructions nécessaires au traitement d'image par rapport à l'utilisation du mode SIMD. Cette réduction du nombre d'instructions pour un traitement donné réduit la fréquence de fonctionnement du circuit et donc sa consommation électrique.

Le mode Multi-SIMD est particulièrement adapté au traitement brut et plus généralement lorsque des prédicats `IF/ELSE`, et `CASE` engendrent des segments de code de longueur importante. Deux implémentations qu'il convient de choisir à la conception du circuit en fonction du budget en surface silicium et des fonctionnalités attendues, sont proposées. Par défaut, la version dite simplifiée adaptée au traitement d'images brutes est utilisée, son surcoût par rapport à une structure SIMD classique étant minime par rapport au gain en nombre d'instructions à exécuter par pixel.

4.6 L'unité de communication

Comme cela a été expliqué dans le chapitre précédent, l'architecture *eISP* est composée d'un ensemble de tuiles de calcul. Ces tuiles sont interconnectées entre elles par un bus TDMA qui leur permet de communiquer via un ou plusieurs canaux. Ce bus permet aussi la synchronisation des tuiles entre elles et rend leur enchaînement paramétrable.

L'unité de communication a un rôle central au niveau de l'architecture, puisque toutes les données à traiter proviennent du flux. Comme les tuiles de calcul traitent les données en temps réel, l'ensemble des modules de la tuile de calcul sont orchestrés par le module de communication. Cette section est divisée en deux parties, la première décrit comment configurer le bus, et notamment ses fréquences de fonctionnement, et la seconde présente l'implémentation du module de communication rendant chaque tuile autonome. Ce module se situe à l'interface entre les tuiles et le bus. Comme chaque tuile fonctionne à sa fréquence qui lui est propre, et que le bus aussi, il permet d'assurer la transmission des données entre des composants qui sont dans des domaines d'horloges différents.

4.6.1 Dimensionnement du bus

Nous avons vu en section 3 que le bus proposé permet de supporter les différents enchaînements de tuiles de calcul susceptibles d'être utilisés. Nous avons aussi vu que la configuration de l'enchaînement des tuiles est réalisée par l'écriture d'un registre permettant de sélectionner le slot temporel correspondant au résultat d'une autre ressource (tuile de calcul, capteur etc.).

Chaque tuile de calcul se voit proposer un espace de temps dans lequel elle peut lire ou écrire, comme cela a été décrit dans le chapitre 3. Afin de déterminer la fréquence de fonctionnement du bus Bus_{Clk}, et donc le nombre de slots disponibles pour l'écriture de données résultant de traitements de tuiles de calcul, il est nécessaire de connaître leurs fréquences de fonctionnement F_i, et la fréquence de l'horloge pixel Px_{Clk}. A partir de ces paramètres, le nombre de slots maximum disponibles sur le bus peut être obtenu par l'équation 4.10 ainsi que la fréquence de fonctionnement maximale du bus par l'équation 4.11. La Figure 4.13 montre un exemple de ou trois horloges (F_1,F_2 et F_3) donnent lieu à trois slots temporels.

$$Nb_{Slots} = \left\lfloor \frac{\min(F_i)}{Px_{Clk}} \right\rfloor \tag{4.10}$$

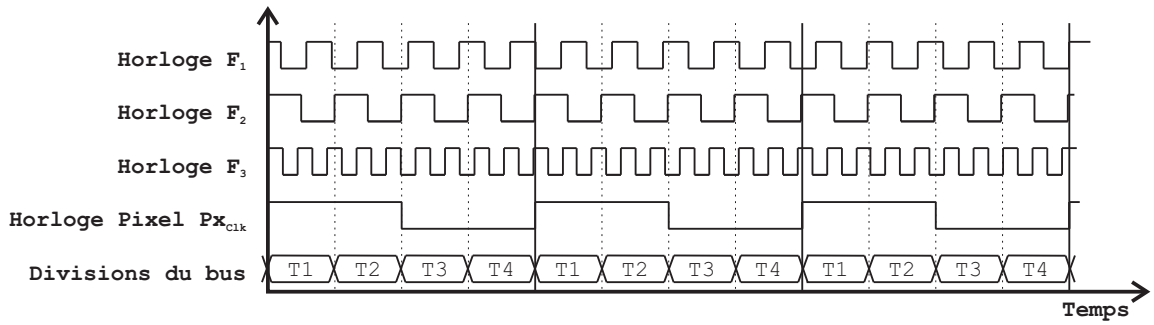


FIG. 4.13: DÉTERMINATION GRAPHIQUE DU NOMBRE DE SLOTS DU BUS TDMA EN FONCTION DES FRÉQUENCES DES TUILES DE CALCUL F_1 À F_3.

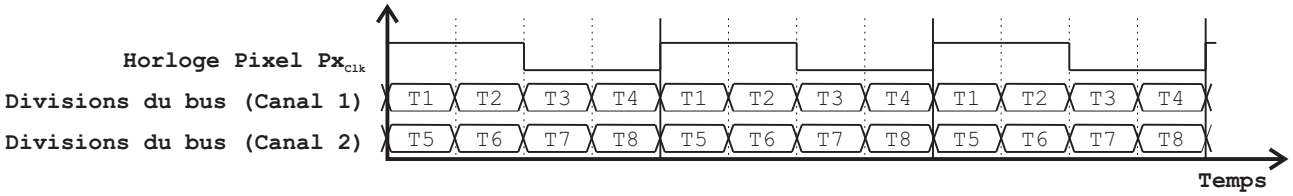


FIG. 4.14: EXEMPLE DE CHRONOGRAMME POUR UN BUS TDMA UTILISANT DEUX CANAUX DE COMMUNICATION.

$$Bus_{Clk} = NbSlots \times Px_{Clk} = NbSlots \times \left\lfloor \frac{\min(F_i)}{Px_{Clk}} \right\rfloor \tag{4.11}$$

Pour connecter plus de tuiles de calcul qu'il n'y a de slots disponibles, il est possible de multiplier le nombre de canaux de transmission du même bus comme en Figure 4.14. Il est ainsi possible de transférer plusieurs valeurs en parallèle durant le même cycle d'horloge du bus TDMA. Le nombre de canaux à utiliser en fonction du nombre de tuiles (*NbTuiles*) est obtenu par l'équation 4.12. Il faut tenir compte des composants d'entrées-sorties (le capteur par exemple). Une autre possibilité pour augmenter le débit du bus est d'augmenter sa fréquence. Cette alternative nécessite l'augmentation des fréquences de fonctionnement des tuiles par rapport à celle de l'horloge pixel, ce qui impacte la consommation électrique.

Il est également envisageable de mettre en œuvre des stratégies d'interconnexions comme le présente la Figure 4.15. Dans cet exemple, la tuile de calcul numéro 2 permet de communiquer aussi bien sur le bus numéro 1 que le bus numéro 2. Il est envisageable de mixer les différentes méthodes proposées ici afin d'adapter au mieux l'architecture aux contraintes de surface et de consommation. Enfin, pour limiter la consommation électrique, les tuiles de calcul sur lesquelles aucun traitement n'est porté peuvent être désactivées.

$$NbCanaux = \left\lceil \frac{NbTuiles}{NbSlots} \right\rceil = \left\lceil \frac{NbTuiles}{\left\lfloor \frac{\min(F_i)}{Px_{Clk}} \right\rfloor} \right\rceil \tag{4.12}$$

4.6.2 Implémentation

L'unité de communication intégrée à la tuile de calcul, décrit par la Figure 4.16, permet de configurer le slot de lecture et d'écrire les résultats sur un ou plusieurs slots. Lors de la programmation de l'architecture, le

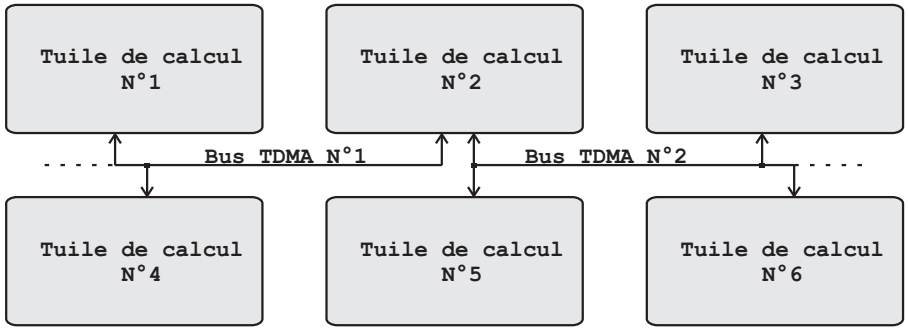


FIG. 4.15: EXEMPLE DE STRATÉGIES DE CONNEXION DES TUILES DE CALCUL.

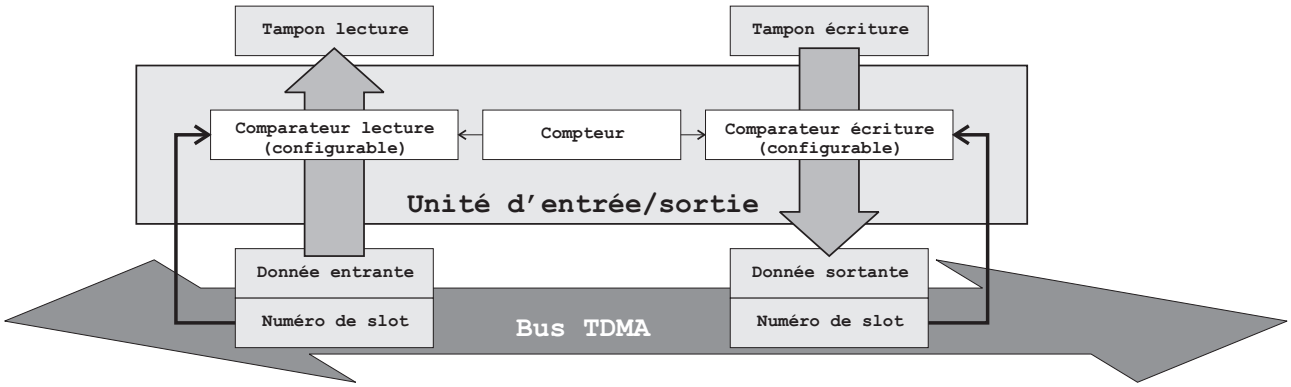


FIG. 4.16: MODULE D'ENTRÉE SORTIE PERMETTANT DE LIRE ET ÉCRIRE SUR LE BUS TDMA.

numéro de ces slots est défini par l'utilisateur en fonction de ses besoins, avec pour conséquence de modifier l'ordre dans lequel s'enchaînent les traitements qui sont programmés sur les tuiles. Afin de permettre cette flexibilité, nous avons conçu les modules de communication tels qu'une tuile ne puisse lire qu'une seule donnée entrante par slot temporel. Plusieurs tuiles peuvent lire le même slot, et donc traiter les mêmes données d'entrées avec des programmes différents. Autrement dit, une tuile ne traite qu'un seul mot de donnée à la cadence de l'horloge pixel. Par contre, une tuile peut écrire sur plusieurs slots et notamment sur plusieurs canaux. Cette fonctionnalité permet à des tuiles de transmettre leurs résultats à d'autres tuiles de calcul alors qu'elles sont placées sur différents canaux, comme dans l'exemple de la Figure 4.15. La vérification des écritures multiples sur le même slot repose sur les outils logiciel de configuration de l'architecture réduisant ainsi la complexité architecturale de l'unité de contrôle, et par là même la surface silicium et la consommation électrique.

4.6.2.1 Accès au données du bus

Lorsqu'une tuile doit lire une donnée transmise sur un des canaux dans un slot donné, l'utilisateur écrit ces valeurs dans les registres correspondants. Ces registres commandent un multiplexeur afin de connecter le canal à lire au signal d'entrée du module de communication. A chaque front de l'horloge du bus, un compteur est incrémenté, et remis à zéro lorsque le nombre de slots maximum est atteint. La valeur de ce compteur est comparée à celle du registre du numéro de slot à lire. Lorsque les deux valeurs sont égales, la valeur sur le bus, considérée alors comme donnée entrante est recopiée sur le registre de lecture, qui est l'entrée du gestionnaire de voisinages. La procédure utilisée pour l'écriture de données sur le bus est identique que celle utilisée pour la lecture.

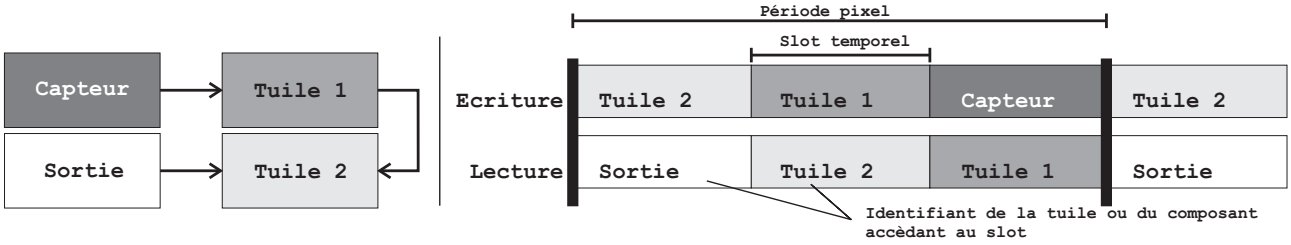


FIG. 4.17: EXEMPLE SCHÉMATISANT L'ORDONNANCEMENT DES ACCÈS AU NIVEAU DU BUS TDMA, À GAUCHE LA CONFIGURATION D'ENCHAÎNEMENT VOULUE, À DROITE L'ACTIVITÉ DES COMPOSANTS SUR CHAQUE SLOT DISPONIBLE PENDANT UNE PÉRIODE PIXEL.

La Figure 4.17 résume comment quatre composants, deux tuiles de calcul, un capteur et un module de sortie accèdent au bus. Le temps est représenté de gauche à droite. On peut voir que le premier slot est occupé par le résultat de la seconde tuile de calcul, c'est ce slot que le module de sortie lit. Le second slot est occupé par le résultat du calcul de la première tuile que la seconde tuile lit, et enfin le troisième slot est occupé par la donnée écrite par le capteur que la première tuile lit.

Conclusion sur l'unité de communication

Le module de communication que chaque tuile de calcul intègre joue un rôle central au niveau de l'architecture. Il maintient synchronisé l'ensemble des tuiles de calcul grâce à l'horloge du bus. Il définit l'ordre avec lequel les tuiles sont enchaînées les unes aux autres, et sert d'interface entre différents modules ou PIs dédiées. Au niveau de la tuile de calcul, le module de communication alimente le gestionnaire de voisinages en données. Enfin, il transmet les résultats du traitement à d'autres tuiles de calcul, ou à d'autres modules.

Nous avons vu que l'implémentation de ce module se révèle relativement simple au regard des autres éléments architecturaux. La particularité de son écriture réside essentiellement dans le fait qu'il permet de passer du domaine d'horloge du bus à celui de la tuile.

4.7 Intégration d'extensions à l'architecture

L'architecture de calcul *eISP* a été conçue pour pouvoir être facilement réduite, en limitant ses ressources au minimum nécessaire à l'exécution des algorithmes, ou bien étendue par l'ajout de ressources supplémentaires. La première méthode consiste à augmenter le nombre de tuiles de calcul et de processeurs que chaque tuile intègre. Si cette solution permet d'offrir un niveau de flexibilité élevé, elle reste coûteuse en surface silicium au regard des performances des composants dédiés.

Trois niveau d'intégration d'extensions ont été introduits à l'architecture *eISP*. D'abord au niveau architecture, les intégrateurs ont souvent développé leurs propres PIs de traitement qu'ils peuvent placer à la place d'une tuile de calcul. Pour cela, il leur suffit d'utiliser le module de communication comme interface avec le bus TDMA. La PI est alors vue comme une tuile de calcul autonome. Nous ne développerons pas plus ce niveau d'intégration. Le second, au niveau de la tuile de calcul, permet l'ajout d'unités fonctionnelles directement dans le chemin de données sans que cela n'ai d'influence sur la flexibilité de l'architecture. C'est l'objet de la première partie de cette section. Enfin, le dernier niveau d'intégration correspond à celui du processeur de calcul.

D'abord par l'ajout d'opérateurs supplémentaires, ensuite par l'utilisation du plan mémoire comme moyen de communication. C'est l'objet de la seconde partie de cette section.

4.7.1 Introduction d'unités fonctionnelles au niveau de la tuile de calcul

De nombreux traitements consistent simplement à exécuter des fonctions sur la valeur d'un seul pixel, dont les résultats sont réutilisés ensuite par les différents processeurs pour les traitements dédiés. C'est pourquoi nous avons introduit des unités fonctionnelles directement dans le chemin de données de la tuile, c'est à dire, au niveau du flux en entrée, et au niveau du flux en sortie. Ces unités peuvent être composées d'une UAL reconfigurable pour des corrections de gain, de luminosité par exemple, ou de LUTs typiquement utilisées pour une correction gamma par exemple. Il peut aussi s'agir d'unités reconfigurables. La Figure 4.18a présente une tuile de calcul intégrant de telles unités fonctionnelles qui sont conçues pour réaliser $y = a \times x + b$ suivie d'une opération stockée en Look-Up Table (LUT).

Comme la cadence des données à traiter est celle de la fréquence pixel (52 MHz en HD 1080p), plusieurs de ces unités peuvent être enchaînées pour réaliser des traitements en plusieurs cycles. La durée du traitement, tant que la contrainte temps réel est respectée, n'impacte que la latence globale du système.

Des instructions spécifiques à la configuration de ces unités fonctionnelles sont introduites et sont traitées directement par l'unité de contrôle, elles ne sont donc pas transmises aux processeurs de la tuile de calcul. Si ces unités fonctionnelles sont des composants reconfigurables qui nécessitent un délai de reconfiguration, il est nécessaire de le faire pendant l'initialisation du système ou pendant les délais d'intertrames.

4.7.2 Introduction d'extensions au niveau du processeur de calcul

L'extension des fonctionnalités du processeur de calcul peut être réalisée à deux niveaux. Le premier par l'ajout d'opérateurs supplémentaires au niveau de l'unité d'exécution, le second par l'ajout de coprocesseurs.

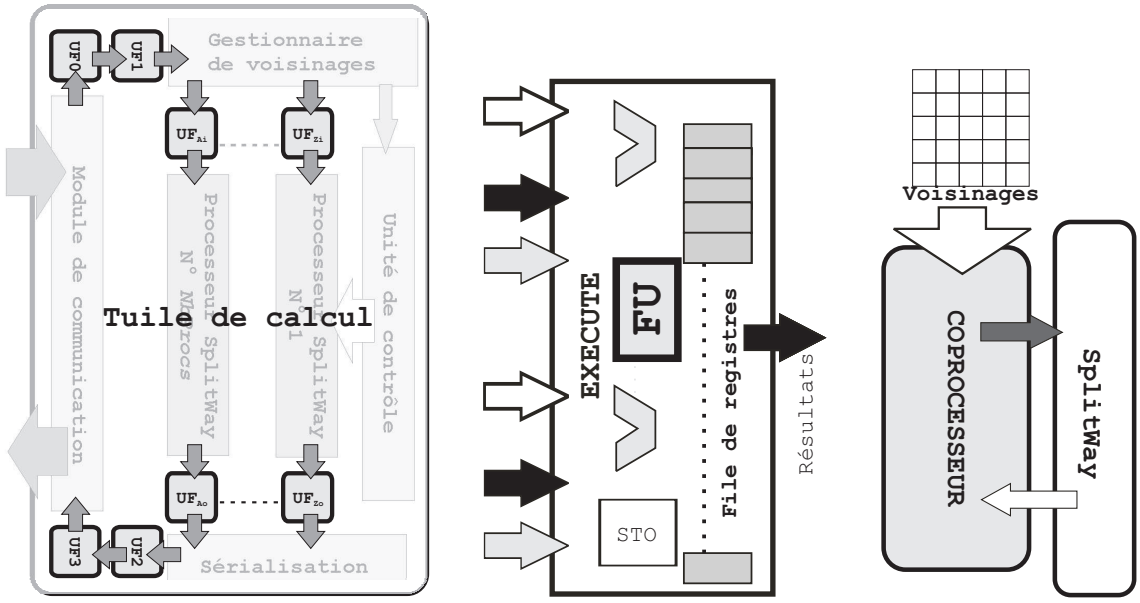
4.7.2.1 Extension du jeu d'opérateurs

Des extensions peuvent être implémentées en tant qu'opérateurs du processeur *SplitWay* comme en Figure 4.18b et bénéficier de l'infrastructure d'accès aux opérandes et à la file de registres du processeur. Ces extensions doivent s'exécuter en un cycle processeur, ou tout au moins leur exécution être divisible en séquences d'un cycle processeur, auquel cas le programmeur doit connaître son temps d'exécution. Ce mode d'extension est parfaitement adapté pour des opérations simples comme une MAC, un *log* ou une division.

4.7.2.2 Ajout de coprocesseurs

Des accélérateurs plus complexes peuvent nécessiter plusieurs cycles pour l'exécution d'un traitement ou ne pas être prévus pour s'intégrer au jeu d'opérateurs, par exemple en raison d'une alimentation particulière en données, issues du gestionnaire de voisinages.

Leur intégration est donc réalisée entre chaque processeur *SplitWay* et le gestionnaire de voisinages, comme le schématise la Figure 4.18c. D'une part le processeur *SplitWay* communique avec son ou ses coprocesseur(s) par l'intermédiaire de l'espace d'adressage de son plan mémoire, d'autre part, le coprocesseur peut être connecté, si nécessaire au gestionnaire de voisinages pour accéder directement aux valeurs des pixels à manipuler, ou encore à d'autres éléments de l'architecture.



(a) Unités fonctionnelles dans le flux de données.

(b) Unités fonctionnelles en tant qu'opérateur.

(c) Coprocesseur connecté à *SplitWay*.

FIG. 4.18: EXEMPLE D'INTÉGRATIONS DE COMPOSANTS DÉDIÉS AU SEIN DES TUILES DE CALCUL, (a) UFS INTÉGRÉES AU FLUX DE DONNÉES, (b) UNITÉ FONCTIONNELLE ACCESSIBLE EN TANT QU'OPÉRATEUR AU SEIN DE L'UNITÉ EXECUTE DU PROCESSEUR, (c) COPROCESSEUR ALIMENTÉ EN DONNÉES PAR LE GESTIONNAIRE DE VOISINAGES ET/OU PAR LE PROCESSEUR VIA L'ESPACE D'ADRESSAGE DU PLAN MÉMOIRE.

Il apparaît que l'utilisation de coprocesseurs induit une forte augmentation de la surface silicium pour des fonctions qui sont dediées. Notons qu'il peut là encore s'agir d'une unité reconfigurable.

Conclusion sur l'intégration de ressources dédiées à l'architecture

Cette section a permis de voir les différents niveaux d'extensions que l'architecture *eISP* peut supporter. Des composants dédiés peuvent s'ajouter ou remplacer certaines tuiles de calcul, ce qui permet aux intégrateurs d'utiliser directement leurs PIs en leur adjoignant le module de communication. Des opérations peuvent être introduites directement dans le flux de pixels au niveau de chaque tuile de calcul, ce qui permet de décharger les processeurs des opérations régulières au niveau pixel, comme la correction gamma ou des corrections de gain. Ces deux premiers modes d'extension n'ont pas d'impact sur la programmabilité des différentes tuiles de calcul.

Le niveau supplémentaire consiste à étendre le jeu d'instructions en ajoutant des opérateurs aux processeurs, ces opérateurs bénéficient alors de l'accès complet aux ressources du processeur. Enfin des coprocesseurs peuvent être ajoutés aux processeurs qui leur sont accessibles par le plan mémoire. Leur alimentation en données peut être réalisée par connexion directe avec le gestionnaire de voisinages.

Si l'utilisation d'extensions dédiées permet d'augmenter drastiquement la capacité de calcul de l'architecture, elle induit aussi l'augmentation de sa surface silicium et de sa consommation électrique. Cette augmentation peut être acceptable si elle n'implique pas une trop grande spécialisation de certaines tuiles de calcul pour des classes de traitement données. Aussi est-il essentiel de déterminer avec précision le choix de chaque accélérateur à utiliser pour ne pas s'éloigner de la flexibilité initiale de l'architecture.

Conclusion

Ce chapitre a détaillé la conception de l'architecture *eISP* dans son ensemble, ainsi que des différents éléments qui la compose. Cette architecture de calcul programmable permet le traitement de flux de données en temps réel, et peut être dimensionnée à la conception du circuit en fonction des besoins de l'utilisateur. Elle est construite autour d'un ensemble de tuiles de calcul, schématisé en Figure 4.19, qui sont interconnectées entre elles par un bus TDMA spécialement conçu.

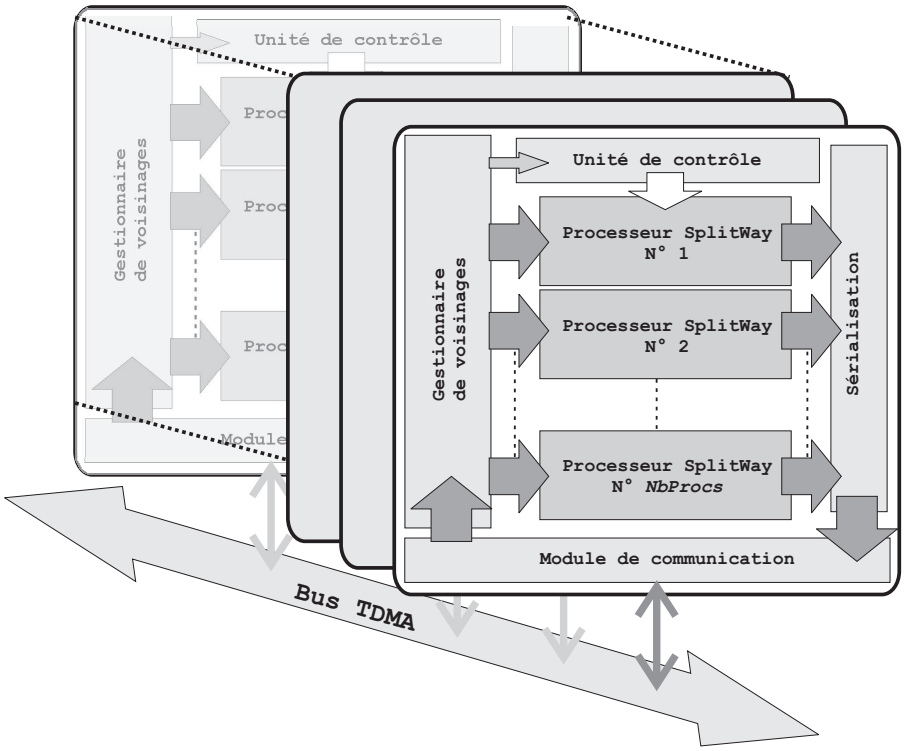


FIG. 4.19: REPRÉSENTATION DE L'ARCHITECTURE DANS SON ENSEMBLE, PLUSIEURS TUILES DE CALCUL QUI PEUVENT ÊTRE HÉTÉROGÈNES SONT INTERCONNECTÉES ENTRE-ELLES.

Afin de maximiser le nombre d'opérations par pixel pouvant être exécutées, différents mécanismes sont mis en œuvre. Comme la surface silicium et la consommation électrique sont deux éléments essentiels de la conception de l'architecture *eISP*, différentes techniques ont été mises en œuvre pour réduire la surface silicium et la consommation électrique de l'architecture et de chaque composant qui la constitue. Les différentes formes de parallélisme extraites lors de l'analyse algorithmique sont traduites d'un point de vue architectural. Les différents éléments de l'architecture sont conçus pour limiter d'abord la surface silicium, ensuite la consommation, tout en assurant les performances attendues.

Le parallélisme de tâche est exploité en enchaînant les tuiles de calcul, qui exécutent chacune des traitements différents. Il est aussi exploité en séparant les tâches d'accès aux données, de contrôle et de calcul, mais aussi d'autres opérations comme l'extraction des métadonnées, ou l'utilisation d'éventuels accélérateurs dédiés. Ces derniers peuvent être intégrés à différents niveaux de la tuile de calcul. Ainsi, le programmeur décrit uniquement les cœurs de boucles correspondant aux équations des filtres qu'il doit porter, sans se soucier de la

manipulation des données. La parallélisme spatial inhérent au traitement d'image est exploité en traitant plusieurs pixels simultanément. Enfin le parallélisme au niveau instruction l'est grâce à l'utilisation de processeurs VLIW deux voies.

L'accès aux données, qui représente la part la plus importante des ressources de calcul requises, est géré à plusieurs niveaux, les processeurs de calcul sont déchargés de ces ressources. D'abord, une hiérarchie mémoire prépare en temps masqué l'ensemble des données à manipuler au sein de la tuile. Il s'agit du gestionnaire de voisinages, qui, comme son nom l'indique, organise les données de l'image sous forme de voisinages. Ceux-ci sont directement accessibles par le processeur, et/ou d'éventuelles PIs dédiées. Ensuite, Le processeur de calcul possède un jeu d'instructions orthogonal qui lui permet d'utiliser n'importe quelle source de données en tant qu'opérande. Enfin, l'utilisation de données composites au niveau de la tuile de calcul dont le format est paramétrable n'induit pas de surcoût algorithmique.

Les ressources programmables de la tuile de calcul reposent sur les processeurs *SplitWay* regroupés pour fonctionner en mode Multi-SIMD. Ce mode de fonctionnement permet de réduire le nombre d'instructions nécessaires au traitement des images brutes, mais aussi des programmes pour lesquels des prédicats de type `IF/ELSE/CASE` sont utilisés et que le mode SIMD n'exécute pas de manière optimale. Toutes les instructions qui sont implémentées traitent des données entières signées et sont exécutées en un seul cycle processeur. Le jeu d'opérations, qui peut être étendu, n'est pas spécifique au traitement d'image, permettant ainsi l'utilisation du processeur *SplitWay* pour tout type de traitement en mode flux de données.

L'unité de contrôle couplée avec le gestionnaire de voisinages assure le fonctionnement des processeurs en mode Multi-SIMD. En mode SIMD, les mêmes instructions du programme sont transmises à l'ensemble des processeurs dès qu'un événement comme l'arrivée de pixels à traiter le commande, tandis qu'en mode Multi-SIMD différents segments de code sont transmis à différents processeurs. Cette fonctionnalité permet d'adapter efficacement le programme aux données, et notamment au traitement d'images brutes constituées d'alternances de pixels de type différents.

Enfin, une unité de communication est intégrée aux tuiles de calcul pour leur permettre de communiquer entre elles sur le bus TDMA ou avec d'autres composants comme un capteur ou un processeur. Les valeurs des pixels sont donc extraites du flux et transmises au gestionnaire de voisinages, tandis que les résultats des processeurs permettent de régénérer un flux de pixels pour l'envoyer à d'autres tuiles de calcul. Pour plus de flexibilité, l'enchaînement des tuiles de calcul est configurable, et par conséquent celui des traitements qui sont programmés dessus. L'unité de communication sépare en outre les domaines d'horloges des tuiles et du bus puisque chaque tuile fonctionne à une fréquence adaptée au traitement qu'elle doit réaliser.

CHAPITRE 5

Génération et validation de l'architecture

Où sont caractérisées les surfaces silicium et les consommations électriques de l'architecture eISP.
Où est montré comment déterminer le nombre de tuiles de calcul et de processeurs de l'architecture en respectant les contraintes initialement fixées.
Où est expliqué comment une chaîne de traitement d'image est portée sur l'architecture eISP.
Où l'architecture eISP est comparée à l'état de l'art.

Introduction

LE CHAPITRE 3 a permis de déterminer les ressources de calcul nécessaires à l'exécution de la chaîne d'amélioration d'image en aval du capteur. Le chapitre 4 a détaillé la conception de l'architecture *eISP* composant par composant, et notamment celle du processeur *SplitWay* qui est utilisé en mode Multi-SIMD au sein de tuiles de calcul programmables. La description comportementale de cette architecture et de ses composants en langage *SystemC* a permis d'en valider le fonctionnement. L'architecture a ensuite fait l'objet d'une description VHDL synthétisable afin d'obtenir des tuiles de calcul programmables dont le fonctionnement en technologie ASIC TSMC 65 nm a été validé après synthèse. Leur nombre, ainsi que leur contenu (nombre de processeurs, éléments de mémorisation, taille maximale des voisinages etc.) doivent être déterminés lors la création du circuit en fonction des contraintes d'intégration. Si ces choix impactent la capacité de calcul et la flexibilité du système, ils impactent aussi la surface silicium et la consommation électrique. En effet, à surface silicium équivalente, plus la surface dédiée aux éléments de mémorisation est importante et plus celle dédiée aux ressources de calcul sera réduite. Il est aussi possible d'augmenter la flexibilité de l'architecture en intervenant par exemple sur la diversité des opérateurs ou sur le volume de ressources de mémorisation.

La Section 5.1 réalise une estimation des surfaces et des consommations électriques que différentes instances de l'architecture *eISP* peuvent prendre. Pour cela il est d'abord nécessaire d'identifier les différents paramètres permettant d'intervenir sur la flexibilité, la capacité de calcul, la surface silicium et la consommation électrique de l'architecture. Un générateur automatique de l'architecture est mis en place pour différentes valeurs de ces paramètres, ce qui permet l'exploration de plusieurs configurations possibles (taille des chemins de données, taille des éléments mémoire, nombre de registres, etc.). A partir des configurations sélectionnées, les surfaces et les consommations électriques sont estimées par synthèse et simulation de différentes instances de l'architecture.

La Section 5.2 présente une évaluation des performances de l'architecture *eISP* en fonction des différentes configurations. Une approche est proposée pour prendre en compte les ressources des fonctionnalités de chaque composant de la tuile de calcul. Enfin, la capacité de calcul effectivement disponible pour le programmeur est présentée en fonction des surfaces silicium et des consommations obtenues.

Après avoir expliqué par l'exemple comment un programme peut être porté sur une tuile de calcul, la section 5.3 présente *Vid'eISP.1*, une instance de l'architecture *eISP* suffisamment flexible et capable de supporter les chaînes d'amélioration vidéo qui ont été présentées dans le chapitre 1. Cette instance est conçue à partir d'une démarche Adéquation Algorithme-Architecture, qui tient compte des résultats obtenus en 5.2 ainsi que des

spécificités des algorithmes. Le portage de la chaîne complète de traitement sur *Vid'eISP.1* est ensuite présenté. Pour démontrer la flexibilité du modèle architecturale *eISP*, deux instances supplémentaires sont présentées dont la surface est de l'ordre du millimètre carré. La première est destinée à privilégier la capacité de calcul, l'autre la flexibilité de programmation.

Enfin, avant de conclure, la dernière Section 5.4 présente une comparaison de l'architecture *eISP* avec les différentes solutions proposées pour le traitement du signal introduites dans le chapitre 2.

5.1 Estimation des surfaces et de la consommation

Nous avons vu que l'architecture *eISP* est constituée de tuiles de calcul programmables. Ces tuiles de calcul, interconnectées entre elles par un bus TDMA, permettent de traiter des flux de données, typiquement des vidéos. Chaque tuile, grâce aux processeurs *SplitWay* qu'elles intègrent, est programmable et présente une capacité de calcul qui dépend du nombre de processeurs et de leur fréquence de fonctionnement. D'une manière générale, la surface et la consommation de l'architecture dépend donc du nombre de tuiles utilisées dans l'architecture ainsi que des éléments qui les constituent. Les paramètres de ces composants ont un impact sur la flexibilité globale de l'architecture. Fixons par exemple le chemin de données des processeurs à 8 bits, la dynamique des calculs possibles sera moindre que s'il est fixé à 16 ou 24 bits rendant ainsi difficile le portage de certains traitements. Par contre, la surface silicium des processeurs étant inférieure, un plus grand nombre d'entre eux pourra être intégré au sein des tuiles de calcul. La capacité de calcul globale s'en trouve ainsi augmentée. Il est même possible, dans certains cas, que ce surcroît de ressources programmables pallie au manque de flexibilité globale de la structure.

5.1.1 Paramètres influant sur la surface et la consommation de l'architecture

Cette sous-section présente différents paramètres ayant une influence directe sur la surface et la consommation de l'architecture *eISP*. Sont rappelés les impacts sur la flexibilité, la surface silicium et la consommation électrique, de ces paramètres pour les différents éléments qui composent l'architecture.

5.1.1.1 Les processeurs *SplitWay*

Les processeurs *SplitWay* possèdent une file de registres dont le nombre, ainsi que la taille, ont une influence directe sur la surface utilisée et sur la flexibilité de programmation du processeur. Tout comme la taille des registres, la taille des opérateurs, et donc du chemin de données du processeur, influent sur la surface, la consommation et la flexibilité de programmation. Il en va de même pour le nombre et la nature des d'opérateurs utilisés. Tous ces paramètres conditionnent directement la taille du processeur élémentaire, et notamment celle de l'étage de décodage. Celui-ci permet de préparer tous les opérandes en accédant à leurs valeurs, et notamment celles issues du gestionnaire de voisinages L'utilisation d'une mémoire de travail augmente la surface silicium du processeur, mais aussi sa flexibilité puisqu'il est possible de l'utiliser pour remplacer certaines opérations complexes par des LUTs pré-calculées. Autrement dit, les choix en termes de taille du chemin de données, nombre de registres et taille de la mémoire de travail ont une influence, que l'on pressent majeure, sur la surface silicium utilisée par le processeur, mais aussi sur sa consommation et sa flexibilité. La fréquence de fonctionnement des processeurs et la nature des opérations utilisées par le programme sont responsables d'une part importante de la consommation électrique d'un processeur.

5.1.1.2 Surface et consommation des gestionnaires de voisinages

Chaque tuile de calcul intègre un gestionnaire de voisinages qui est conçu pour mettre en forme et apporter aux processeurs *SplitWay* des voisinages dont la taille peut aller jusqu'à $N_{Max} \times M_{Max}$ pixels, définie à la conception du circuit, N_{Max} étant le nombre de lignes du voisinage, et M_{Max} le nombre de colonnes. Le programmeur peut ensuite accéder aux voisinages des pixels comme il le souhaite. La surface occupée par le gestionnaire de voisinages est constituée des N_{Max} mémoires lignes, et des registres de voisinages, dont le nombre dépend essentiellement du nombre de processeurs de la tuile de calcul et du nombre de lignes N_{Max} du voisinage. De plus, la taille en bits des mots de données a un impact majeur sur la surface du gestionnaire de voisinages, or elle peut être de 8 bits, ou encore de 24 bits pour un pixel composite rouge, vert, bleu ou plus si le programmeur souhaite intégrer des métadonnées. La surface et la consommation du gestionnaire de voisinages dépendent essentiellement du nombre de lignes maximales N_{Max} du voisinage et de la taille des mots de données que la tuile peut supporter ainsi que du nombre de processeurs qu'elle intègre.

5.1.1.3 Surface et consommation de l'unité de contrôle

Chaque tuile de calcul comprend une unité de contrôle. Elle transmet aux processeurs les instructions qu'ils doivent exécuter. Comme l'architecture peut fonctionner en mode Multi-SIMD, la même instruction est transmise à différents groupes de processeurs d'une tuile de calcul. Il faut donc autant de mémoires programme qu'il y a de groupes de processeurs pouvant fonctionner en Multi-SIMD, le pire cas revient à un mode Multiple Instruction Multiple Data (MIMD), où il y a autant de mémoires programme que de processeurs. La taille des mémoires programme ont aussi une influence sur la surface de la tuile. Les exemples qui suivent sont présentés avec deux mémoires programme de 256 mots de 32 bits chacun, ce qui correspond à la gestion simplifiée des métadonnées. Nous préconisons l'utilisation quatre mémoires programme d'au moins 256 mots d'instruction.

5.1.1.4 Le module de communication

Chaque tuile de calcul intègre un module de communication, il a pour fonction de lire le flux de pixels à traiter sur un ou plusieurs canaux TDMA. Il a aussi pour fonction de transmettre à un canal un flux de pixel qu'une autre tuile pourra ensuite traiter. Ce bus permet la configuration de l'ordre dans lequel les tuiles se transmettent les données ainsi que la synchronisation des tuiles de calcul entre elles. L'utilisation de plusieurs bus permet d'assurer le regroupement des tuiles. Ainsi des tuiles de calcul homogènes, c'est à dire présentant les même propriétés en termes de nombre de processeurs, ressources de mémorisation etc., peuvent directement être chaînées les unes avec les autres. Le nombre de canaux que le module de communication est en mesure de gérer ainsi que la taille des données à transmettre a une influence sur sa surface et sa consommation électrique puisque la taille des registres tampon en dépend.

5.1.1.5 Les extensions à l'architecture

Nous avons vu en Section 4.7, que différents types d'extensions peuvent être aisément introduits à plusieurs niveaux de l'architecture. D'une manière générale, l'utilisation de telles extensions augmente la surface et la consommation de l'architecture. Les composants introduits au niveau de la chaîne de traitement, à la place d'une tuile de calcul ont un impact faible sur la souplesse de l'architecture. Leur surface et leur consommation ne dépendent que de leur conception. Les extensions intégrées au niveau de la tuile ne sont instanciées qu'une seule fois, mais la flexibilité introduite à ce niveau reste limitée et ne dépend d'aucun autre paramètre que des fonctions qu'elles réalisent. Les extensions introduites auprès des processeurs (différents niveaux sont possibles) augmentent la flexibilité de programmation en introduisant des fonctionnalités supplémentaires. Leur surface et leur consommation dépendent de leur taux d'utilisation et du nombre de processeurs intégrés à la

tuile, puisqu'elles leurs sont adjointes. Une étude exhaustive des extensions possibles à l'architecture *eISP* dépasse le cadre de cette étude tant les possibilités sont importantes.

5.1.2 Génération automatique de l'architecture

Nous venons de le voir, de nombreux paramètres peuvent influer non seulement la surface silicium et la consommation électrique des éléments qui constituent l'architecture *eISP*, mais aussi sa flexibilité et ses performances. Afin d'en tenir compte précisément et d'étudier les différentes organisations architecturales qui en découlent, nous générons automatiquement le modèle VHDL synthétisable des différentes tuiles de calcul qui composent l'architecture à partir des valeurs des paramètres à étudier. Pour cela, nous avons conçu et réalisé l'outil *geneSP* (pour Signal Processor Generator) qui génère une description VHDL synthétisable de l'architecture *eISP*, optimisée pour différentes cibles (FPGA, ASIC) mais aussi d'adapter l'assembleur correspondant. Le choix des technologies cibles permet d'utiliser des librairies de composants pour la génération des opérateurs et ainsi appeler l'outil de synthèse adapté. Par exemple, la librairie *Synospsy DesignWare* [Synopsys 2009] est utilisée pour l'ensemble des opérateurs arithmétiques et logiques lorsque le circuit est destiné à une synthèse ASIC. Les outils de synthèse et d'analyse en consommation peuvent ensuite être lancés. Quelques-uns des paramètres identifiés au niveau de tuiles de calcul sont les suivants :

- le nombre de processeurs ;
- le nombre de regroupement Multi-SIMD ;
- la taille des mémoires programme ;
- la taille du voisinage maximal accessible par les processeurs ;
- la taille des images ;
- la taille des données à transmettre entre les tuiles (et donc à mémoriser dans les gestionnaires de voisinages) ;
- le nombre maximal de champs de métadonnées et leur utilisation ou non.

De plus, des paramètres des processeurs *SplitWay* de chacune des tuiles de calcul peuvent être aussi configurées, dont une liste exhaustive suit :

- la taille du chemin de données du processeur ;
- le nombre d'opérateurs à intégrer aux processeurs ;
- le choix des opérateurs à intégrer aux processeurs (voir 4.3.1 page 89)
- la taille de la mémoire de travail des processeurs ;
- la taille du chemin de données des processeurs ;
- le nombre de drapeaux intégrés aux processeurs élémentaires ;
- le nombre de registres de la file de registres ;
- la taille des registres de la file de registres ;

Le code VHDL ainsi généré est ensuite synthétisé avec l'outil *Synopsys Design-Compiler-Ultra* [Arvind 2008], pour une technologie d'intégration cible (typiquement TSMC 65 nm) et une fréquence maximale de fonctionnement. Cette méthode permet d'obtenir le modèle Resistor Transistor Logic (RTL) de l'architecture qui peut ensuite être exploité pour déterminer la surface et la consommation du circuit.

5.1.3 Obtention des surfaces

A partir des différentes descriptions VHDL obtenues par le générateur automatique, l'architecture est synthétisée composant par composant, puis dans son ensemble pour différentes configurations dépendant des pa-

ramètres précédemment cités, mais aussi pour une fréquence de fonctionnement maximale et pour une technologie d'intégration donnée (ASIC TSMC 65 nm etc., FPGA etc.). Toutes les surfaces et les consommations électriques présentées au long de ce chapitre sont obtenues pour la technologie TSMC 65 nm basse consommation.

5.1.3.1 Synthèse ASIC de l'architecture

La synthèse de l'architecture génère son modèle RTL qui interconnecte les différentes cellules standards (dénommées cellules dans la suite de ce chapitre) disponibles dans la technologie d'intégration cible, elles s'apparentent à des portes logiques ou à des ensembles de portes logiques. Pour cela, les outils de synthèse (*Synopsys Design-Compiler*,*Cadence SoC Encounter, BuilderGates* etc.) s'appuient sur la librairie de cellules fournies par le fondeur TSMC pour une technologie donnée (par exemple TSMC 65 nm [TSMC 2007]) et sur la description comportementale VHDL du circuit. Outre la disponibilité à notre laboratoire de la suite d'outils de synthèse *Synopsys* et notamment *Design-Compiler-Ultra*, sa capacité à prendre en compte les contraintes de consommation du circuit lors de la synthèse en fait un outil adapté pour la synthèse au niveau cellules de notre architecture.

La surface du circuit obtenue est la somme des surfaces des cellules, tandis que sa fréquence maximale de fonctionnement est donnée par le chemin critique. En effet, le temps de traversée de chaque cellule est donné par le fondeur, le chemin critique est obtenu en décomptant leur nombre traversées entre deux fronts d'horloge, et ce pour chacun des signaux du circuit. Ses choix impactent donc la fréquence maximale qu'il est possible d'atteindre ainsi que la surface silicium utilisée. L'outil de synthèse est en mesure d'utiliser différentes techniques d'agencement des cellules pour réaliser une même fonction logique en tenant compte des contraintes qui lui sont spécifiées. Ces choix impactent donc la fréquence maximale qu'il est possible d'atteindre ainsi que la surface silicium utilisée. Dans notre cas, nous rechercherons une surface silicium minimale pour une contrainte en fréquence que nous nous sommes fixé à 400 MHz soit un chemin critique de moins de 2.5 ns. Par expérience, la synthèse à 400 MHz nous assure un fonctionnement du circuit à 250 MHz, toutefois une étude approfondie de ce point pourrait permettre d'optimiser les résultats obtenus, elle dépasse le champ de ce manuscrit.

5.1.3.2 Résultats de l'étude en surface post-synthèse

Les résultats obtenus à partir des synthèses de différentes instances de l'architecture *eISP* générées pour différentes valeurs des paramètres listés précédemment permettent d'en étudier les impacts sur la surface des tuiles de calcul. Le niveau de granularité que nous avons choisi pour représenter les résultats est d'une part le processeur *SplitWay* et d'autre part la tuile de calcul. En effet, l'architecture *eISP* étant un arrangement de différentes tuiles de calcul qui peuvent être hétérogènes comme homogènes, il n'est pas pertinent de présenter la surface de l'ensemble de l'architecture à ce stade de l'étude. De plus, l'interconnexion entre les tuiles induit peu de surface supplémentaire puisque les unités de communication entre tuiles et bus TDMA sont incluses dans les surfaces présentées. La surface dédiée aux chemins d'interconnexion ne peut pas être estimée lors de la synthèse, mais nous savons à partir des chapitres 3 et 4, que ces bus sont composés de plusieurs canaux, typiquement de 8 à 24 signaux, associés à une horloge.

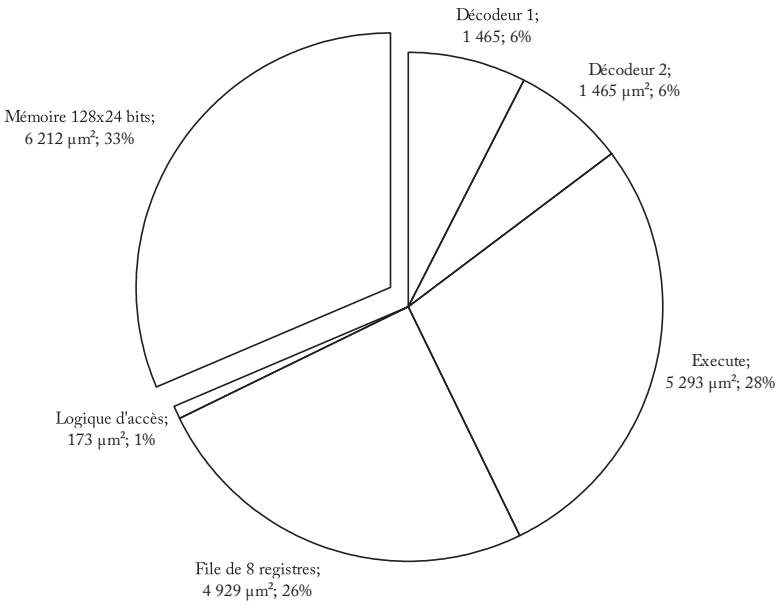


FIG. 5.1: RÉPARTITION DE LA SURFACE ENTRE LES DIFFÉRENTS ÉLÉMENTS DU PROCESSEUR *SplitWay* 24 BITS.

Le processeur *SplitWay* Le processeur *SplitWay* étant un des éléments de base de l'architecture *eISP*, il est essentiel d'en estimer précisément la surface. Suite à l'analyse algorithmique, nous avons choisi d'utiliser un chemin de données de 24 bits afin d'assurer une dynamique importante lors des calculs. Toutefois cette même analyse nous a permis de voir que 16 bits de chemin de données étaient suffisants pour de nombreux traitements. C'est pourquoi nous avons choisi de rendre paramétrable la largeur du chemin de données. Les surfaces du processeur *SplitWay* sont présentées en Table 5.1 pour un chemin de données de 24 bits et de 16 bits. La Figure 5.1 présente la répartition de la surface des différents éléments du processeur *SplitWay*, pour une instance 24 bits. On peut voir sur cette figure que la surface de la file de registres et des opérateurs de l'unité d'exécution représente la plus grande partie de la surface du processeur après la mémoire de travail. Cette mémoire de travail n'étant pas toujours utilisée, il est possible de la supprimer ou de la réduire au sein de certaines tuiles de calcul et ainsi réduire la surface silicium utilisée par le processeur. Celle-ci peut, par exemple, être mise à profit pour augmenter le nombre de processeurs intégrés, ou encore intégrer des mémoires de travail de taille plus importante au sein d'autres tuiles de calcul.

TAB. 5.1: SURFACE DES DIFFÉRENTS ÉLÉMENTS DU PROCESSEUR *SplitWay* EN FONCTION DE LA TAILLE DU CHEMIN DE DONNÉES ET OPÉRATEURS (ICI 16 ET 24 BITS) EN TECHNOLOGIE TSMC 65 NM.

Composant	**Chemin de données 24 bits**	**Chemin de données 16 bits**
A File de registres (8 registres)	49 29 μm^2	2 175 μm^2
Unité d'exécution	52 93 μm^2	1 410 μm^2
Décodeur $\times$ 2	2$\times$1 465 μm^2	2$\times$467 μm^2
Logique diverse	173 μm^2	106 μm^2
Surface total :	**13 325 μm^2**	**4 625 μm^2**
Complexité kGates :	**6,2 kGates**	**3,5 kGates**

Les gestionnaires de voisinages La Figure 5.2 présente la surface silicium dédiée aux processeurs *SplitWay* par rapport à la surface dédiée au gestionnaire de voisinages. Différents cas sont couverts par cette figure puisque sont représentés les cas où la tuile de calcul intègre des processeurs *SplitWay* 24 bits sans mémoire de travail ainsi que le cas où ces processeurs possèdent une mémoire de 256 mots de 24 bits. Cette surface est normalisée par rapport à une tuile de calcul de deux processeurs sans gestionnaire de voisinages (dimensionné à 1$\times$1). C'est sans surprise que la surface utilisée par les processeurs associés à une mémoire de travail est nettement plus élevée que celle utilisée par ceux n'ayant pas de mémoire de travail. Deux cas sont traités

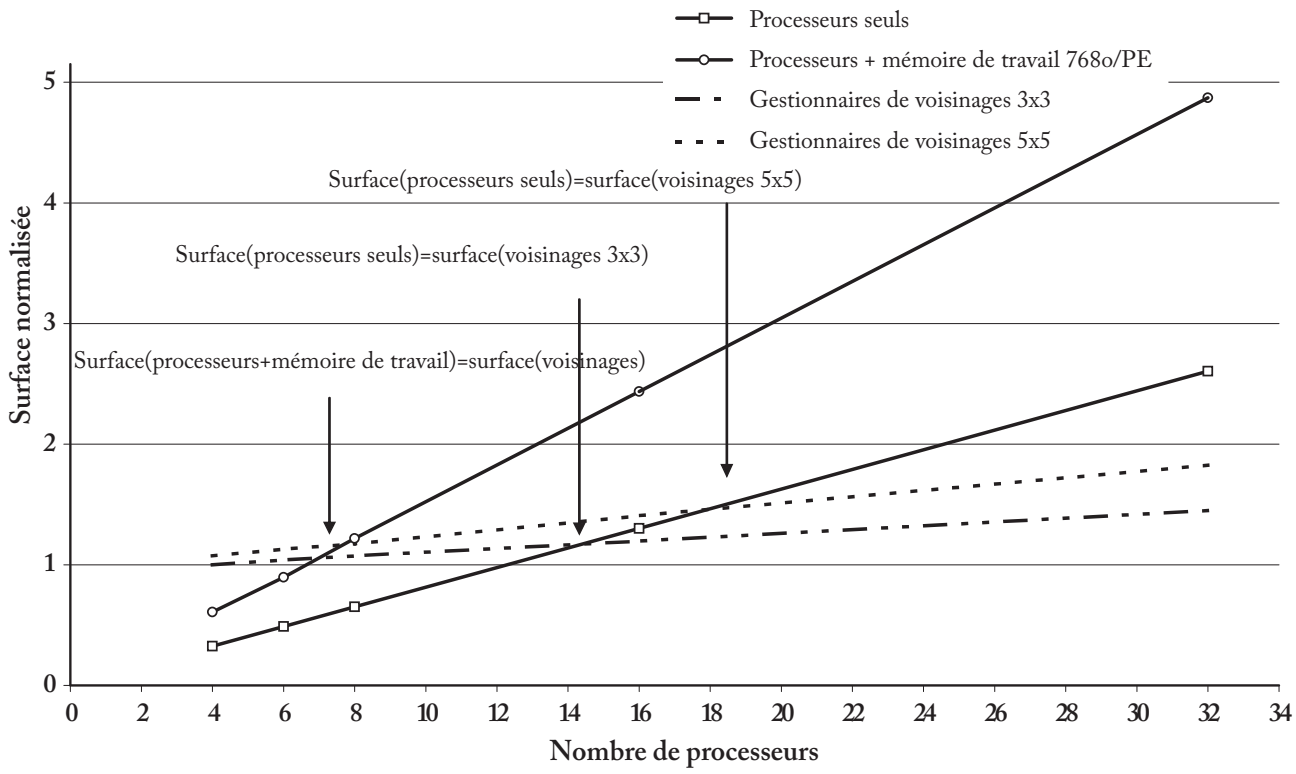


FIG. 5.2: SURFACE DÉDIÉE AUX PROCESSEURS *SplitWay* ET SURFACE DÉDIÉE AU GESTIONNAIRE DE VOISINAGES EN FONCTION DU NOMBRE DE PROCESSEURS POUR UNE TUILE DE CALCUL DONNÉE. LE NOMBRE DE PROCESSEURS À PARTIR DUQUEL LA SURFACE DE CALCUL DEVIENT PLUS IMPORTANTE QUE LA SURFACE DES ÉLÉMENTS DE MÉMORISATION (EN FONCTION DE LA TAILLE DES VOISINAGES GÉRÉS) SONT INDIQUÉS – PROCESSEURS *SplitWay* 24 BITS ET MOTS DE DONNÉES DE 8 BITS, SURFACE NORMALISÉE PAR RAPPORT À UNE TUILE DE CALCUL DE DEUX PROCESSEURS SANS GESTIONNAIRE DE VOISINAGES.

concernant la surface allouée aux gestionnaires de voisinages, le premier est le support de voisinages 3×3 et le second, le support de voisinages 5×5. Pour des petites tailles de voisinages, la surface silicium croît linéairement avec la taille des voisinages, ce qui est cohérent, puisqu'on sait que l'élément faisant varier la taille du gestionnaire de voisinages est le nombre de lignes et le nombre de processeurs à alimenter. Notons toutefois que si la taille du voisinage devient importante, le recouvrement le sera aussi, ainsi que le nombre de ports des registres de voisinages et la cellule utilisée également (ou bien elle sera dupliquée). Cette figure permet, à titre complémentaire, de mettre en évidence le nombre de processeurs à partir duquel la surface silicium dédiée au calcul devient supérieure à la surface dédiée à la gestion des voisinages.

Taille des mots de données et surface du gestionnaire de voisinages La Figure 5.3 présente la surface, d'une tuile de calcul en fonction de la taille des mots mémorisés pour un nombre de processeurs fixé. Cette surface est normalisée par rapport à une tuile de calcul qui ne nécessite pas de mémorisation de voisinages (dimensionné à 1×1). En observant cette figure, il semble plus « rentable » de gérer des mots de grande taille, par exemple composite (rouge, vert et bleu) sur la même tuile de calcul.

Surface d'une tuile de calcul Afin de déterminer le nombre de tuiles, il est nécessaire de déterminer la surface totale des tuiles de calcul en fonction de la taille des voisinages à traiter, ainsi que du nombre de

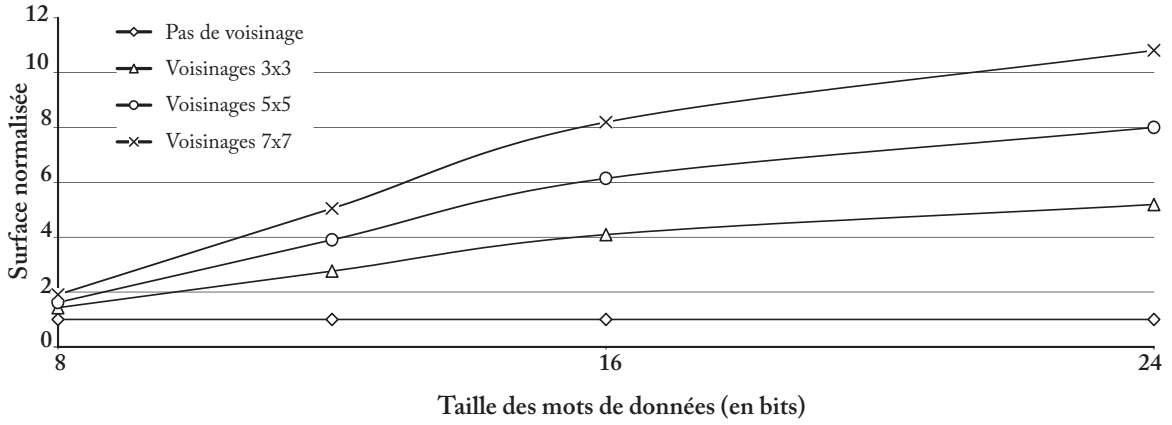


FIG. 5.3: ÉVOLUTION DE LA SURFACE NORMALISÉE DE L'ARCHITECTURE EN FONCTION DE LA TAILLE DE MOTS À TRAITER ET DE LA TAILLE DU VOISINAGE $N \times M$.

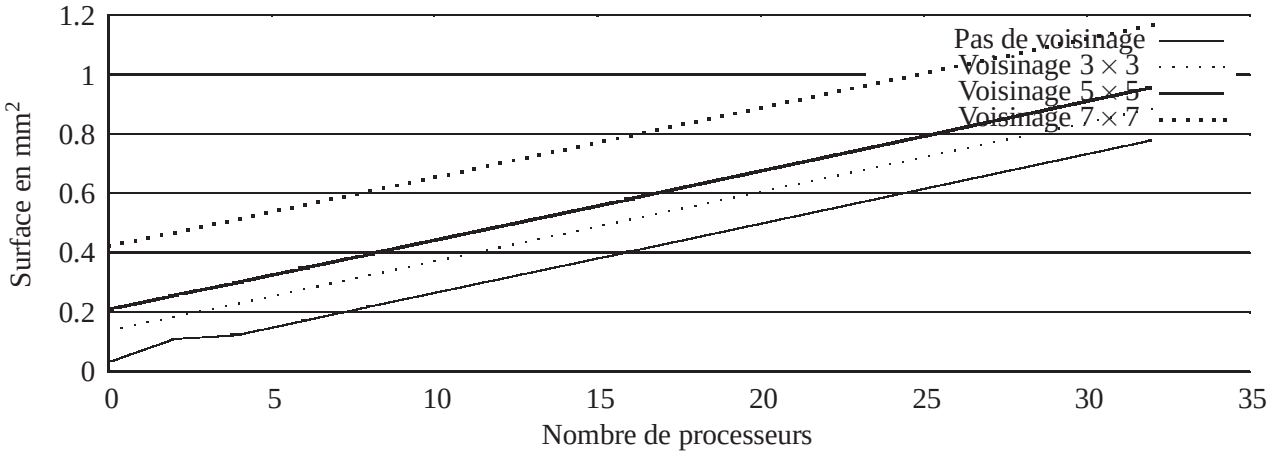


FIG. 5.4: ÉVOLUTION DE LA SURFACE D'UNE TUILE DE CALCUL EN FONCTION DU NOMBRE DE PROCESSEURS *SplitWay* ET DE LA TAILLE DES VOISINAGES $N \times M$ GÉRÉS – PROCESSEURS *SplitWay* 24 BITS INTÉGRANT 256 MOTS DE 24 BITS DE MÉMOIRE DE TRAVAIL ET CAPABLE DE TRAITER DES MOTS DE 24 BITS (SOIT 3 PIXELS DE 8 BITS).

processeurs qui la compose. C'est l'objet de la figure 5.4 qui présente la surface silicium utilisée pour des tuiles de calcul allant jusqu'à 32 processeurs *SplitWay* 24 bits auxquels sont associés une mémoire de travail de 256 mots de 24 bits. Les tuiles sont synthétisées pour des données issues du flux de pixel de 24 bits, ce qui permet de traiter et de stocker des triplets de 8 bits par exemple. la figure 5.3 présente les surfaces de tuiles de calcul en fonction de la taille des données issues du flux à stocker. On peut voir qu'une tuile traitant des mots de 8 bits présente une surface très nettement inférieure à celles traitant des mots de grande taille. C'est pourquoi il est crucial d'adapter la taille des gestionnaires de voisinages aux données à traiter, puisqu'ils représentent une part importante de la surface des tuiles de calcul.

Unité de contrôle et module de communication L'étude en surface a consacré une place importante au gestionnaire de voisinages et aux processeurs *SplitWay*. La surface de l'unité de contrôle capable de supporter deux groupes de processeurs *SplitWay* en mode Multi-SIMD est de 3 674 μm^2 associée à deux mémoires programme de 256 mots de 32 bits de près de 7 000 μm^2 chacune, soit un total de moins 20 000 μm^2. La surface

du module de communication capable de communiquer sur deux canaux TDMA varie en fonction de la taille des données à transmettre (8 à 24 bits) entre moins de 2 500μm^2 et et 55 000μm^2 en technologie TSMC 65 nm. La surface totale des éléments de contrôle et de communication est donc inférieure à 25 0000μm^2 par tuile de calcul, ce qui correspond approximativement à la surface d'un processeur *SplitWay* 24 bits. Nous retiendrons donc que les variations de surface du module de communication et de l'unité de contrôle représentent un impact mineur sur la surface totale d'une tuile de calcul.

Placement-routage Comme la synthèse de l'architecture ne permet pas de tenir compte de la distance qui sépare deux cellules standards ni de la surface nécessaire à leur interconnexion, nous avons choisi d'affiner les résultats de la synthèse par une étape de placement-routage de l'ensemble de l'architecture pour une configuration donnée. Cette étape permet d'obtenir la surface réelle du composant en tenant compte des spécificités de la technologie cible, dans notre cas 65 nm. Cette surface connue, il est ainsi possible de déterminer avec précision le chemin critique puisque la longueur des connexions entre cellules est connue. Le placement-routage a été réalisé sur une tuile de calcul de 6 processeurs. Cette étude a essentiellement permis de mettre en évidence le surcoût d'interconnexion qui atteint en effet 31 % pour l'unité d'exécution du processeur *SplitWay*. La Figure 5.5 représente le résultat du placement routage d'une tuile de 6 processeurs *SplitWay* 24 bits, chacun associé à une mémoire de travail 256 mots de 24 bits et d'un gestionnaire de voisinage 3×3, la surface totale obtenue 0,26 mm^2 pour 272 MHz en technologie 65 nm. Le surcoût moyen lié au placement-routage pour l'ensemble de l'architecture par rapport aux résultats post-synthèse est de 23 % pour une augmentation du chemin critique de 32 %. Autrement dit, l'architecture synthétisée pour une fréquence maximale de 400 MHz présente une fréquence de fonctionnement après placement routage de 272 MHz. L'estimation en consommation électrique a pu être étudiée avec les outils de la suite *Cadence Soc Encounter*.

5.1.4 Caractérisation des consommations

La puissance électrique P_{Tot} d'un circuit est obtenue par le calcul de la consommation statique P_S de l'équation 5.1, et de la consommation dynamique [Texas Instrument 1996] comme le décrit l'équation 5.5 [Piguet 2004, Texas Instrument 1996]. Lors de la conception de ce circuit, nous avons pu intervenir sur le nombre de transistors tandis que l'utilisation d'une technologie d'intégration basse consommation permet de réduire les courants de fuite I_{fuite}. La Puissance dynamique dissipée, équation 5.4, dépend de deux termes, la puissance dissipée lors des transitions des transistors du circuit (P_T de l'équation 5.2) et de la puissance dissipée par les charges capacitives du circuit (P_L de l'équation 5.3). La possibilité d'adapter la fréquence de la tuile de calcul permet d'intervenir sur les paramètres relatifs aux fréquences des signaux $F_{in/out}$ (que nous approximons par F qui est la fréquence de fonctionnement des tuiles). De plus, lors de de la conception de l'architecture nous avons cherché à limiter le paramètre A_{sw} qui correspond aux changement d'état des transistors, c'est tout particulièrement le cas avec le gestionnaire de voisinages. Les capacités internes C_{pd} sont réduites en limitant le nombre de transistors. Les capacités « externes » aux transistors C_L sont réduites en limitant le nombre de transistors et leur interconnexion. Dans les deux cas (C_{pd} et C_L), l'utilisation des technologie basse consommation permet de limiter leurs valeurs.

$$P_S = V_{CC} \times \sum I_{fuite} \tag{5.1}$$

$$P_T = C_{pd} \times F_{in} \times {V_{CC}}^2 \times \sum A_{sw} \tag{5.2}$$

$$P_L = C_L \times {V_{CC}}^2 \times F_{out} \times \sum A_{sw} \tag{5.3}$$

$$P_D = P_L + P_T \tag{5.4}$$

$$P_{Tot} = P_S + P_D \tag{5.5}$$

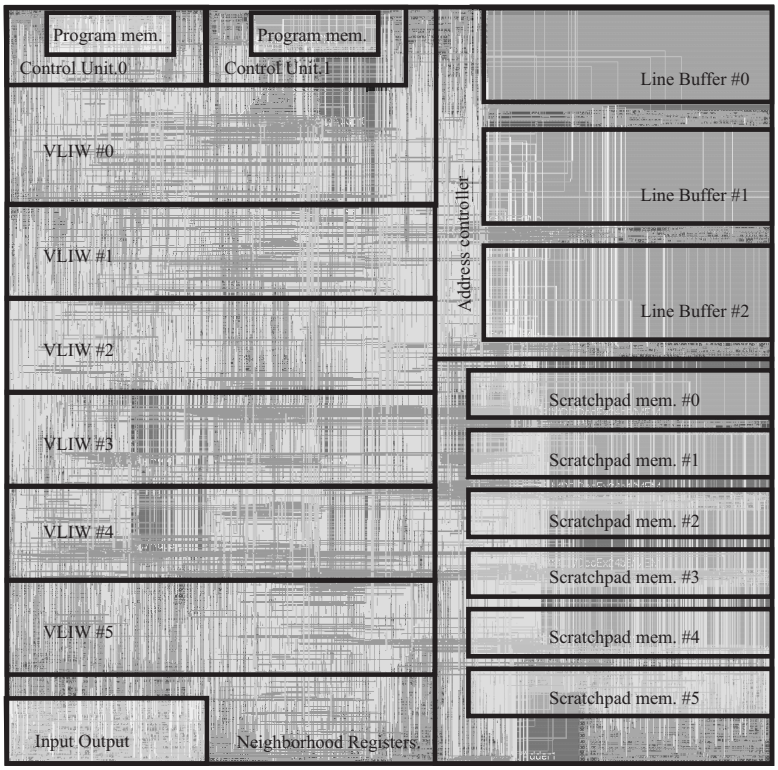


FIG. 5.5: RÉSULTAT DU PLACEMENT ROUTAGE D'UNE TUILE DE 6 PROCESSEURS *SplitWay* 24 BITS, CHACUN ASSOCIÉ À UNE MÉMOIRE DE TRAVAIL 256 MOTS DE 24 BITS ET D'UN GESTIONNAIRE DE VOISINAGE 3×3, LA SURFACE TOTALE OBTENUE 0,26 MM^2 POUR 272 MHZ EN TECHNOLOGIE 65 NM.

À tension d'alimentation fixée, la fréquence de fonctionnement du circuit a une influence directe sur la consommation électrique dynamique. La fréquence maximale de fonctionnement du circuit en dépend puisqu'elle est liée au délai de propagation des transistors. Ce délai diminue avec la tension, potentiellement on peut envisager l'utilisation de technique de variation de la tension d'alimentation. Or, il est généralement nécessaire d'élever la tension d'alimentation avec la fréquence de fonctionnement du circuit, ce qui augmente la consommation électrique du circuit. L'utilisation de techniques permettant de faire varier la fréquence du circuit ont été utilisées en vue d'implémenter des techniques de variation de la tension d'alimentation des composants du circuit. Ces techniques tendent aujourd'hui à être matures, elles pourront être étudiées dans un second temps.

Par ailleurs, les données à traiter et le programme à exécuter ont une influence sur le nombre de transitions du circuit. Par exemple, si toutes les données à traiter ont la même valeur, le nombre de transitions est réduit. De plus, si les opérateurs appelés sont constitués de peu de transistors, leur consommation dynamique restera faible. Utiliser de techniques de codages adaptées [Ringgenberg 2006] permet de réduire cette contribution, ce que nous n'avons pas choisi de faire pour pouvoir utiliser des opérateurs arithmétiques et logiques déjà existants. Pour réaliser l'estimation de la consommation électrique de notre circuit, nous nous appuierons sur l'outil *Synopsys PrimePower* [Synopsys 2006a, Synopsys 2006b] qui permet d'obtenir les consommations électroniques pour un composant synthétisé au niveau portes logiques, pour une tension de fonctionnement donnée (dans notre cas, 1,3v).

La consommation statique des cellules standards est connue puisque renseignée par le fondeur. La consommation dynamique peut être estimée par les outils d'analyse à partir des données que le fondeur associe à

chaque élément de sa librairie de cellules standards. Or la synthèse du circuit permet de connaître le taux d'utilisation des cellules standards pour chaque composant de l'architecture. Pour cela, nous avons simulé et enregistré la valeur des signaux pour chacune des entrées et sorties de chacune des cellules standards à partir du modèle post-synthèse, et ce pour le circuit en fonctionnement nominal. A partir de ces enregistrements et des informations associées à chacune des cellules standards par le fondeur, les outils logiciels peuvent estimer leur consommation électrique, et donc, celle du circuit complet. Toutefois cette méthode d'estimation de la consommation électrique nécessite des temps de simulation particulièrement élevés. Pour cette raison, nous avons choisi de réaliser des points de mesure sur un jeu de données réel, d'abord en régime nominal pour des programmes représentatifs de la chaîne de traitements ainsi que pour des programmes faisant intensément appel à des opérateurs gourmands tels que les multiplieurs.

Les mesures de consommation ont été réalisées en appliquant la méthode usuelle [Wu Qifa 2002], qui permet d'obtenir des résultats proches de ceux post-placement-routage, et ce, pour des fréquences de fonctionnement à 133 MHz ainsi qu'à 200 MHz sans variation de la tension d'alimentation pour un taux d'utilisation des processeurs supérieur à 75 %. Dans le premier cas de fonctionnement, en régime dit nominal, la représentation des instructions est conforme à l'étude de la chaîne de traitement d'image telle que vue en 3.1.2.3 (page 54). Dans le second mode de fonctionnement, une convolution est implémentée, qui consiste essentiellement en une succession de multiplications et d'additions sur chacun des pixels d'un voisinage donné, en l'occurrence 3×3.

Par ailleurs, pour la même raison liée aux temps de simulation, seules quelques mesures de la consommation électrique de tuiles de calcul ont été réalisées avec cette méthode. Les tuiles de calcul ont été synthétisées en technologie TSMC 65 nm basse consommation, pour une fréquence cible de 450 MHz avec l'outil de synthèse *Design-Compiler* [Arvind 2008]. Dans notre cas, l'outil Power-Compiler qui est en mesure de synthétiser un circuit afin d'optimiser la consommation du circuit à l'aide des cellules standards prévues à cet effet par le constructeur, mais aussi en appliquant automatiquement les techniques de réduction de la consommation. La consommation électrique des bancs mémoire a été calculée manuellement à partir des données fournies par le fondeur. Les accès à ces bancs ayant été précisément décomptés. La consommation électrique obtenue par analyse du circuit est ensuite vérifiée après placement-routage à l'aide de la suite d'outil *SoC Encounter* de *Cadence* [Cadence DS 2006].

La vérification est basée sur une étude statistique du circuit par rapport aux éléments qui le composent, mais peut être réalisée, comme précédemment à partir de l'enregistrement des signaux après simulation. Cette dernière méthode, plus précise requiert d'avantage de temps. Pour effectuer cette analyse statistique nous considérons un taux d'activité de 0,25. Autrement dit, il est considéré qu'un quart des transistors du circuit changent d'état à chaque cycle d'horloge. Ce taux d'activité correspond à un cas d'utilisation standard mesuré d'une tuile de calcul essentiellement composée de 6 processeurs *SplitWay* 24 bits et sa mémoire de travail de 256 mots de même taille, et d'un gestionnaire de voisinages 3×3.

Résultats de l'étude en consommation La Table 5.2 présente la consommation électrique d'une tuile composée de six processeurs *SplitWay* instanciés pour un chemin de données de 24 bits. Une telle tuile de calcul présente une consommation de 11.50 mW à 133 MHz tandis qu'elle augmente à 21,4 mW lorsque la fréquence passe à 200 MHz. Cette table montre qu'une telle instance du processeur *SplitWay* présente une consommation inférieure à 1 mW lorsque sa fréquence de fonctionnement est inférieure à 133 MHz, et une consommation inférieure à 2 mW lorsqu'elle est à 200MHz. Il faut ajouter la consommation d'éventuelles mémoires de travail intégrées aux processeurs. Ces mémoires consomment entre 0,24 et 0,36 mW supplémentaire par processeur *SplitWay*, selon leur fréquence de fonctionnement et le nombre d'accès qui sont réalisés. Dans cet exemple,

l'ensemble de la consommation du gestionnaire de voisinages représente environ 16% de la consommation totale de la tuile de calcul, alors que sa surface représente souvent la moitié de celle d'architecture.

La Figure 5.6 généralise ces résultats en présentant la consommation d'une tuile de calcul en fonction du nombre de processeurs. L'analyse en consommation n'étant réalisée que pour certaines instances des tuiles de calcul, les résultats sont extrapolés afin de proposer un modèle validé expérimentalement. On peut voir que la consommation électrique est proportionnelle au nombre de processeurs utilisés sur la tuile de calcul, ce qui est cohérent puisqu'ils exécutent tous les mêmes instructions.

Par ailleurs, comme le gestionnaire de voisinages a été conçu pour limiter la consommation électrique, notamment en utilisant des mémoires simple-port fonctionnant à la fréquence de l'horloge pixel, nous avons réalisé une étude sur la variation de la consommation électrique en fonction de la taille du voisinage utilisé. Pour cela nous avons choisi une tuile de calcul intégrant 16 processeurs *SplitWay* 24 bits, ce qui nous place dans le cas où la surface silicium utilisée par les processeurs est au moins égale à la surface utilisée pour la gestion des voisinages.

Afin de vérifier que la variation de consommation est liée au gestionnaire de voisinages associé aux bancs mémoire, nous avons choisi d'exécuter un même programme quelle que soit la taille du voisinage disponible sur la tuile de calcul. Ainsi, la variation de consommation enregistrée correspond bien à la surconsommation du gestionnaire de voisinages à proprement parler. Il est toutefois évident qu'un programme nécessitant l'accès à un voisinage 25×25 exécutera plus d'opérations qu'un programme nécessitant un accès à un voisinage 3×3, ne serait-ce que par les lectures des valeurs des pixels. Par conséquent, l'augmentation de la consommation électrique en sera d'autant plus importante. La Figure 5.7 présente les résultats de cette étude qui confirme le faible impact sur la consommation électrique globale de notre gestionnaire de voisinages sur l'ensemble d'une tuile de calcul. La consommation électrique est normalisée par rapport au cas 1×1, autrement dit sans gestion de voisinage.

TAB. 5.2: RÉSULTATS DE L'ÉTUDE EN CONSOMMATION POUR UNE TUILE DE CALCUL COMPORTANT 6 PROCESSEURS *SplitWay* DE 24 BITS TRAITANT DES MOTS DE DONNÉES 8 BITS EN HD 1080P.

Composant	**Consommation à 133 MHz**		**Consommation à 200 MHz**	
	dynamique	**statique**	**dynamique**	**statique**
6× processeurs *SplitWay*	6×0.84 mW	6×39 μW	6× 1.81 mW	6× 45 μW
Mémoire de travail 256×24 bits	6×0.24 mW	6×9 μW	6× 0.36 mW	6× 9 μW
3× mémoires lignes 2048× 8 bits	3×0.32 mW	48μW	3×0.32 mW	48 μW
Gestionnaire de voisinages	0.98 mW	77 μW	2.21 mW	80 μW
2× unités de contrôle[1]	2×1.17 mW	2×20 μW	2× 2.16 mW	2× 21 μ
Module de communication	0.38 mW	9 μW	0.46 mW	11 μW
Total	11.04 mW	0.46 mW	20.87 mW	0.53 mW
Total (dynamique + statique)	11.50 mW		21.4 mW	

[1] Deux unités assurent un fonctionnement Multi-SIMD. chacune intègre une mémoire programme 256 mots de 24 bits

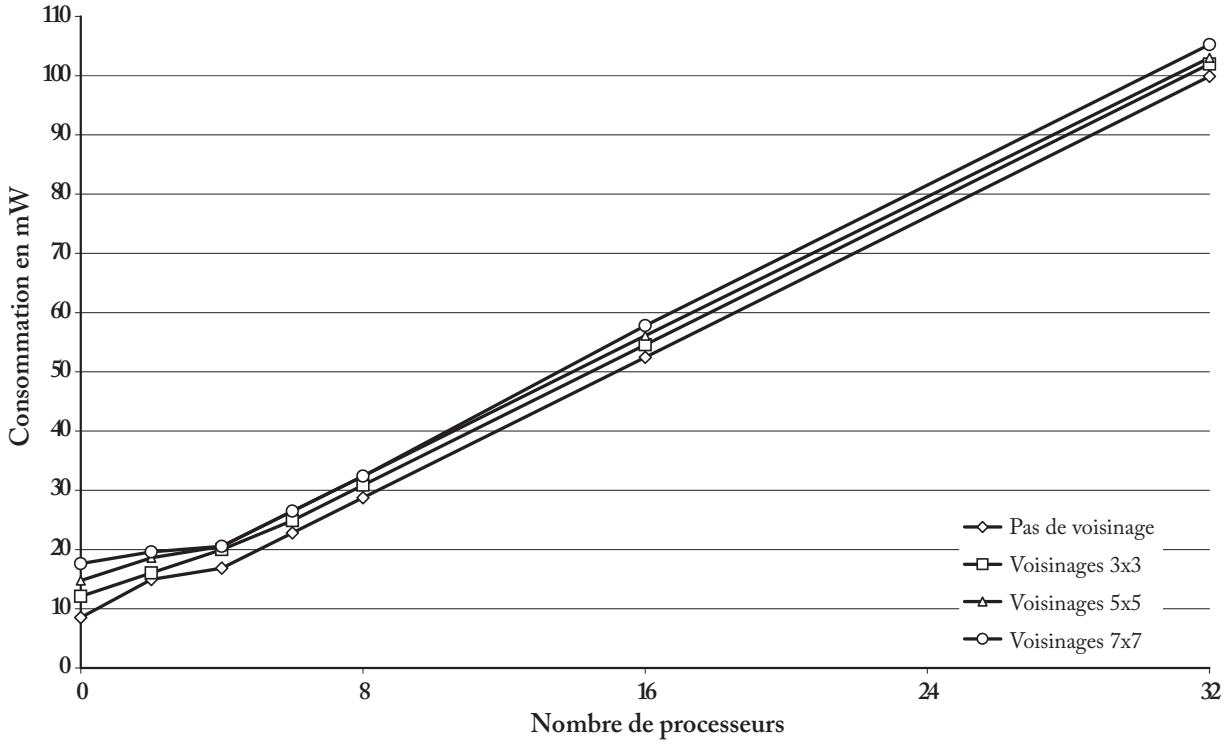


FIG. 5.6: ÉVOLUTION DE LA CONSOMMATION ÉLECTRIQUE EN FONCTION DU NOMBRE DE PROCESSEURS – PROCESSEURS *SplitWay* 24 BITS .

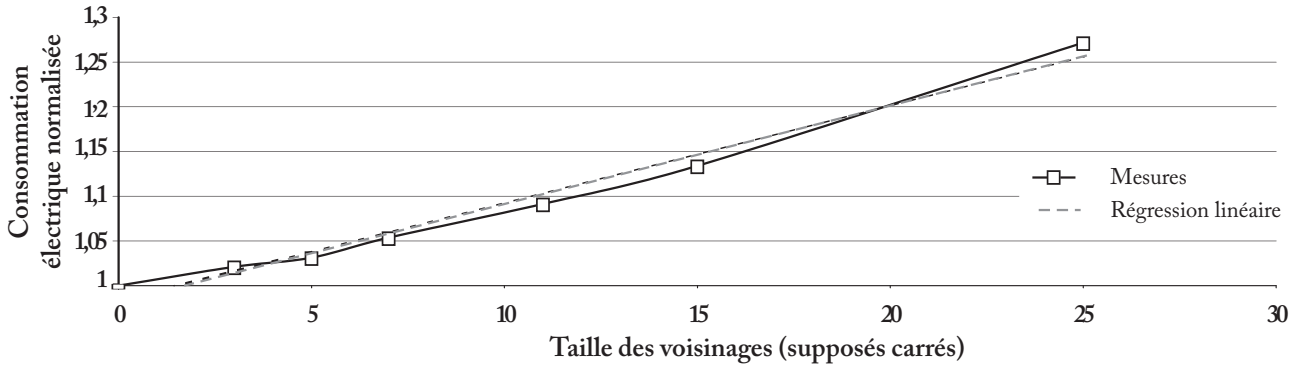


FIG. 5.7: ÉVOLUTION DE LA CONSOMMATION ÉLECTRIQUE NORMALISÉE EN FONCTION DE LA TAILLE DES VOISINAGES.

Conclusion de la caractérisation en consommation

Après avoir présenté les éléments relatifs à l'implémentation de l'architecture dans le chapitre précédent, cette première partie a traité de l'impact sur la surface silicium et la consommation électrique des différents paramètres de l'architecture qui sont fixés à sa création. Une étude sur la surface silicium occupée par l'architecture a été réalisée en technologie TSMC 65 nm et nous a montré que les éléments influant significativement sur la surface des tuiles de calcul sont le nombre de processeurs qu'elles intègrent et le nombre de lignes du gestionnaire de voisinage.

De plus, une étude similaire réalisée sur la consommation électrique de l'architecture montre que la consommation électrique des tuiles qui composent l'architecture *eISP* varie essentiellement en fonction du nombre de processeurs *SplitWay* qu'elles intègrent. D'autre part, la consommation électrique du gestionnaire de voisinages est maintenue sous contrôle et son impact sur la consommation globale de l'ensemble d'une tuile de calcul reste limité à quelques pourcents. Cela est essentiellement lié à la conception du gestionnaire de voisinages, au sein duquel la plupart des changements d'état de ses transistors s'opèrent à la fréquence de l'horloge pixel.

L'étude et l'optimisation en consommation peuvent être améliorées au niveau de l'ensemble de l'architecture, d'une part en utilisant les outils de synthèse basse consommation, en appliquant des techniques de gestion de la consommation au niveau des portes logiques, mais aussi après placement-routage. De nombreuses voies sont alors ouvertes pour l'amélioration de la consommation électrique de l'architecture *eISP*, comme par exemple l'utilisation de méthodes de codage limitant les transitions et adaptant la tension à la fréquence de fonctionnement utilisée. Rappelons que le choix de la fréquence de fonctionnement est réalisé de manière statique par conséquent l'utilisation d'un contrôleur dynamique n'est pas nécessaire.

5.2 Caractérisation des performances de l'architecture

Grâce à l'outil *geneSP* que nous avons développé, les différents éléments qui composent l'architecture *eISP* peuvent être générés selon différents paramètres. Le choix de ces paramètres a un impact direct sur la surface et sur la consommation électrique du circuit, mais aussi sur les performances et la flexibilité de l'architecture *eISP*. Les performances de l'architecture peuvent être évaluées selon plusieurs méthodes, ce qui rend complexe l'estimation de la capacité de calcul réelle de l'architecture *eISP*. Elle peut être évaluée en tenant compte de l'ensemble des opérations réalisées par tous les composants de l'architecture, y compris les éléments dédiés. Toutefois nous préférerons considérer la capacité de calcul effective puisqu'elle correspond aux ressources directement exploitables par le programmeur. Nous avons vu que les choix effectués en termes d'éléments de mémorisation ont un impact direct sur ces mesures. C'est pourquoi, deux métriques sont choisies pour permettre d'évaluer la capacité de calcul, par unité de surface, et par unité de consommation.

5.2.1 Estimation de la capacité de calcul totale

La capacité de calcul « brute » de l'ensemble de l'architecture correspond à la somme des capacités de calcul de chacune des tuiles de calcul. Leur capacité de calcul correspond elle-même à la somme des capacités de calcul des éléments qui les composent.

Les processeurs Les processeurs de calcul *SplitWay* présentent la possibilité d'exécuter deux opérations par cycle d'horloge processeur, ce qui leur confère 2 MOPs/MHz, soit par exemple $\frac{1}{2}$ GOPs avec une fréquence de l'horloge processeur fixée à $Proc_{Clk} = 250\ MHz$. Il faut aussi ajouter les ressources équivalentes à l'utilisation du jeu d'instructions orthogonal. Nous avons vu dans le chapitre 3, que le nombre d'opérations était en

moyenne réduit de 23 %. A partir de ce résultat nous calculons l'accélération potentielle qui en découle, qui est de $\frac{1}{1-0,23} = 1,30$. Comme les tuiles de calcul intègrent *NbProc* processeurs *SplitWay*, la capacité de calcul correspondante à l'ensemble des processeurs est définie par l'équation 5.6 et 5.7 si le gain lié à l'orthogonalité du jeu d'instructions est pris en compte.

$$Res_{Procs} = NbProc \times 2 \times Proc_{Clk} \tag{5.6}$$

$$Res_{ProcsOrtho} = Res_{Proc} \times 1,30 \tag{5.7}$$

Ainsi une tuile de calcul cadencée à 250 MHz intégrant 6 processeurs *SplitWay* développe 3,0 GOPs crête pour sa seule partie dédiée au calcul du traitement d'image tandis que le passage à 16 processeurs permet d'atteindre 8,0 GOPs. La même tuile de 16 processeurs cadencée à 133 MHz développe 4,2 GOPs.

Le gestionnaire de voisinages Le gestionnaire de voisinages est construit autour d'une machine à état configurable associée aux mémoires ligne. Il ne s'agit donc pas d'éléments programmables, mais il est possible d'effectuer le décompte des opérations élémentaires réalisées (lecture en mémoire, incréments, comparaisons etc.) par seconde. En considérant qu'une seule opération est réalisée pour la mise à jour de l'ensemble des registres de voisinages, pour N lignes de voisinage, ce décompte est donné par l'équation pour une horloge pixel fixée à $Pixel_{Clk}$. Cette équation fait apparaître 11 et 7 opérations nécessaires au calcul des positions et des adresses des lignes mémoire, il s'agit principalement de compteurs et de comparaisons.

$$Res_{Gvois_{min}} = Pixel_{Clk} \times (11+7) \times N) \tag{5.8}$$

C'est ainsi que pour un flux vidéo HD cadencé à 52 MPixels par seconde, les ressources de calcul mises en œuvre par le gestionnaire de voisinages devant mettre en forme un voisinage 5×5 peuvent être évaluées à $52 \times (11 + (7 \times 5)) = 2,39$ *GOPs*. Cette évaluation ne prend pas en compte le nombre de processeurs intégrés à la tuile de calcul, c'est pourquoi nous proposons de considérer qu'autant de décalages sont réalisés au niveau des registres de voisinages qu'il y a de *NbProc* processeurs, alors ce décompte nous est donné par l'équation 5.9.

$$Res_{GvoisMax} = Pixel_{Clk} \times (11 + (7 + NbProc) \times N) \tag{5.9}$$

Dans ce cas, les ressources de calcul du gestionnaire de voisinages sont évaluées à $52 \times (11 + (7+16) \times 5)) = 6,55$ *GOPs* lorsque 16 processeurs *SplitWay* sont desservis.

Une autre méthode permettant d'évaluer la capacité de calcul du gestionnaire de voisinages est de considérer que sa présence permet de réduire de moitié les opérations du processeur de calcul, comme nous avons pu l'estimer lors de l'analyse algorithmique. Cette évaluation empirique ne tient pas compte du compte de la structure SIMD de l'architecture et du fait qu'un seul gestionnaire de voisinages est intégré par tuile de calcul quel que soit le nombre de processeurs utilisés.

Par ailleurs, le gestionnaire de voisinages peut être objectivement caractérisé par son débit en lecture, c'est à dire par le nombre de mots qui peuvent être lus simultanément par les composants d'une même tuile de calcul. D'une manière générale, le débit maximal du gestionnaire de voisinages est obtenu par l'équation 5.10, où N_{Max} et M_{Max} sont respectivement la largeur et la hauteur de la fenêtre de voisinages mise à disposition aux composants de la tuile de calcul. La quantité de données mise à disposition aux composants d'une même tuile de calcul peut alors dépasser plusieurs Tbits par seconde.

$$DebitMax_{GVois} = N_{Max} \times M_{Max} \times NbProc \times Proc_{Clk} \times Taille_{Mot} \tag{5.10}$$

Or, les processeurs *SplitWay* peuvent lire et traiter simultanément deux mots issus du gestionnaire de voisinages par cycle d'horloge. Ainsi le débit en lecture en bits par seconde des registres de voisinages est déterminé par l'équation 5.11, où $NbProc$ correspond au nombre de processeurs de la tuile de calcul, $Proc_{Clk}$ leur fréquence de fonctionnement et $Taille_{Mot}$ la taille des mots de données en bits (généralement la taille utilisée pour stocker un pixel ou bien un vecteur de trois pixels)

$$Debit_{GVois} = 2 \times NbProc \times Proc_{Clk} \times Taille_{Mot} \tag{5.11}$$

Par exemple, dans le cas d'une tuile contenant $NbProc = 8$ processeurs cadencés à la fréquence de $Proc_{Clk} = 200\ MHz$ et traitant un flux de données de taille $Taille_{Mot} = 3 \times 8\ bits = 24\ bits$, le débit du gestionnaire de voisinages est de $Debit_{GVois} = 76{,}8\ Gbit/s$. Le gestionnaire de voisinages apparaît donc sous-exploité au regard de sa capacité, c'est pourquoi il peut être envisagé d'associer d'autres composants capables d'accéder à ses données.

L'unité de contrôle L'évaluation des ressources de l'unité de contrôle peut elle aussi être calculée par le décompte des opérations réalisées par la machine à état de contrôle. Quinze opérations sont exécutées par cycle processeur (cadencé à $Proc_{Clk}$) et par groupe de processeurs fonctionnant Multi-SIMD. Or, la version implémentée de l'architecture *eISP* intègre deux groupes de processeurs fonctionnant en mode Multi-SIMD. Dans le cas d'une tuile de calcul Multi-SIMD, les ressources de calcul mises en œuvre par l'unité de contrôle sont définies par l'équation 5.12, et correspondent à 7,5 GOPs à 250 MHz.

$$Res_{UC} = 2 \times 15 \times Proc_{Clk} \tag{5.12}$$

L'unité de communication L'unité de communication permet de lire et d'écrire des flux vidéo sur les $NbCanaux$ canaux du bus TDMA. Ce dernier est cadencé au minimum par l'horloge Bus_{Clk} et réalise l'équivalent de 5 opérations par canal exploité et par cycle d'horloge du bus auxquelles viennent s'ajouter 4 opérations par pixel qui consistent en la lecture et l'écriture des données entrantes et sortantes. Les ressources de calcul équivalentes à l'unité de communication sont données par l'équation 5.13. Dans le cas standard, la fréquence de l'horloge du bus est égale au produit de l'horloge pixel et du nombre de slots $NbSlots$ disponibles : $Bus_{Clk} = Pixel_{Clk} \times NbSlots$.

$$Res_{UComm} = NbCanaux \times 5 \times Pixel_{Clk} \times NbSlots + 4 \times Pixel_{Clk} \tag{5.13}$$

Les ressources ainsi en jeu sont équivalentes à 2,28 GOPs pour un module traitant 2 canaux de 4 slots chacun cadencés par une horloge pixel de 52 MHz correspondant au HD 1080p.

Les extensions dédiées Les extensions dédiées qui peuvent être intégrées au sein de l'architecture de calcul augmentent drastiquement sa capacité de calcul. Il n'est pas possible d'évaluer les performances Res_{Ext} obtenues grâce à leur utilisation puisque celles-ci dépendent de leur nature et de leur nombre.

L'ensemble de la tuile de calcul A partir des ressources de calcul que nous avons estimées pour chacun des composants des tuiles de calcul, il est possible de déterminer les ressources de calcul globales de l'ensemble de l'architecture. Cette dernière est donnée par l'équation 5.14.

$$\sum_{i=\{ProcsOrtho,Gvois_{min},UC,UComm,Ext\}} Res_i \leq Res_{Tuile} \geq \sum_{i=\{ProcsOrtho,Gvois_{Max},UC,UComm,Ext\}} Res_i \tag{5.14}$$

Pour le cas d'une tuile traitant un flux vidéo intégrant 16 processeurs *SplitWay* cadencés à 250 MHz et un gestionnaire de voisinages 5×5, il est possible d'estimer les ressources de calcul équivalentes entre 22,57 et 24,73 GOPs, soit trois fois celles effectivement dédiées à la partie calcul qui est de 8 GOPs. Une telle instance présente une surface d'environ 0,4 mm^2 pour une consommation de 60 mW, ce qui donne une capacité de calcul de 500 MOPs/mW pour plus 70 GOPs/mm^2 à 250 MHz. Ces chiffres sont obtenus essentiellement grâce aux différentes ressources dédiés qui viennent d'être présentées puisque la capacité de calcul des 16 processeurs *SplitWay* représente 8 GOPs. Notons que ces valeurs sont conformes aux résultats de l'analyse algorithmique réalisée au chapitre 3, qui annonce que seulement le tiers des ressources étaient effectivement dédiées au calcul.

5.2.2 Capacité effective de calcul

La sous-section précédente nous a permis d'estimer les ressources de calcul globalement mises en œuvre au sein de l'architecture *eISP*. Comme nous avons choisi de séparer les ressources destinées au calcul effectif du traitement d'image des ressources dédiées à l'accès aux données, nous considérerons dans la suite du document la capacité de calcul effective, autrement dit celle correspondant aux opérations effectivement dédiées au calcul du résultat du filtre. Il s'agit donc uniquement des ressources définies par l'équation 5.6. La capacité de calcul d'une tuile est donc la somme des capacités de calcul des différents processeurs *SplitWay*, qui est obtenue en multipliant le nombre de processeurs par le double de leur fréquence de fonctionnement. Durant le traitement, les données correspondant aux pixels des voisinages ne sont pas accessibles aux processeurs, puisqu'il s'agit du temps de mise à jour des registres de voisinage.

Le nombre d'instructions par cycle permet de déterminer les opérations qu'il est possible d'exécuter pour un traitement donné. Nous avons vu dans les chapitres précédents que la capacité de calcul en instructions par cycle d'une tuile de calcul dépend directement du nombre de processeurs $NbProcs$ de la tuile, de leur fréquence d'horloge $Proc_{Clk}$ et de la cadence d'entrée des pixels $Pixel_{Clk}$. Les Tables 5.3 et 5.4 présentent le nombre d'opérations disponibles pour le traitement de chaque pixel en fonction du nombre de processeurs et de leur fréquence, pour respectivement des flux vidéo HD 720p et HD 1080p. On voit par exemple sur ces tables, que la tuile de calcul intégrant 6 processeurs présentée précédemment en 5.1.4 est capable de délivrer 14 opérations par cycle à 133 MHz et 34 opérations par pixel à 200 MHz, ce qui permet d'exécuter des traitements tels qu'une convolution 3×3.

$$NbCycles = \left(\frac{Proc_{Clk}}{Pixel_{Clk}} \right) \times NbProcs \tag{5.15}$$

TAB. 5.3: NOMBRE D'OPÉRATIONS DISPONIBLES PAR PIXEL EN FONCTION DU NOMBRE DE PROCESSEURS ET DE LA FRÉQUENCE POUR UN FLUX VIDÉO HD 720P À 25 IMAGES PAR SECONDE.

	Nombre de processeurs							
Fréquence de la tuile	**2**	**4**	**6**	**8**	**16**	**24**	**32**	**64**
80 MHz	14	30	46	62	126	190	254	510
133 MHz	22	46	70	94	190	286	382	766
175 MHz	30	62	94	126	254	382	510	1 022
200 MHz	38	78	118	158	318	478	638	1 278
250 MHz	46	94	142	190	382	574	766	1 534
300 MHz	58	118	178	238	478	718	958	1 918

TAB. 5.4: NOMBRE D'OPÉRATIONS DISPONIBLES PAR PIXEL FONCTION DU NOMBRE DE PROCESSEURS DE ET DE LA FRÉQUENCE POUR UN FLUX VIDÉO HD 1080P À 25 IMAGES PAR SECONDE.

	Nombre de processeurs							
Fréquence de la tuile	**2**	**4**	**6**	**8**	**16**	**24**	**32**	**64**
80 MHz	2	6	10	14	30	46	62	126
133 MHz	6	14	22	30	62	94	126	254
175 MHz	10	22	34	46	94	142	190	382
200 MHz	10	22	34	46	94	142	190	382
250 MHz	14	30	46	62	126	190	254	510
300 MHz	18	38	58	78	158	238	318	638

5.2.3 Capacité de calcul par unité de surface

Afin de tenir compte de la surface dans le calcul de performance de l'architecture *eISP*, nous proposons d'évaluer la capacité de calcul par unité de surface. Celle-ci est exprimée en milliard d'opérations par millimètre carré (GOPs/mm^2), et tient compte uniquement des ressources de calcul des processeurs *SplitWay*, autrement dit seule l'équation 5.6 de la page 127 est évaluée pour déterminer les ressources de calcul d'une tuile de calcul. Cette estimation ne tient donc pas compte, ni des éléments de mémorisation, ni des ressources du gestionnaire de voisinages.

Le processeur *SplitWay* Le processeur *SplitWay* présente une surface de 13 325 μm^2 pour une capacité de calcul de $\frac{1}{2}$ GOPS à 250 MHz, ce qui lui confère une capacité de calcul maximale de 37,5 GOPs par mm^2 à 250 MHz. Cette capacité de calcul peut être accrue par l'augmentation de la fréquence, nous avons toutefois vu que 272 MHz était la limite en technologie TSMC 65 nm après placement-routage ce qui lui conférerait 40.8 GOPs par mm^2. La réduction de la taille du chemin de données à 16 bits permet de diminuer significativement la surface silicium du processeur *SplitWay*, et par là même, d'augmenter sa capacité de calcul par unité de surface. Avec 4 625 μm^2 il est alors possible d'atteindre 108,1 GOPs/ mm^2 à 250 MHz et 117,6 GOPs/mm^2 et donc de franchir le seuil des 100 GOPs/mm^2.

L'architecture *eISP* Nous avons vu que différents paramètres impactent directement la surface et la consommation de l'architecture *eISP*, mais aussi la capacité de calcul globale. Parmi ceux-ci, la fréquence de fonctionnement et le nombre de processeurs ont une influence sur la capacité de calcul et la consommation tandis que le nombre de processeurs, et surtout la taille des données traitées et les éléments de mémorisation (gestionnaire de voisinages et mémoires lignes) ont une influence sur la surface silicium. La Figure 5.8 présente la capacité de calcul disponible en milliard d'opérations par seconde en fonction du nombre de processeurs et de la taille des voisinages utilisés. Il apparaît que plus la taille des voisinages est petite, et plus il est possible d'atteindre une capacité de calcul élevée par millimètre carré.

Ces résultats sont donnés pour une fréquence de fonctionnement de 250 MHz, avec des processeurs *SplitWay* de 24 bits de largeur et dans le cas où la taille des éléments de mémorisation est maximisée puisque des mots de 24 bits par pixel sont considérés. Dans ce cas pessimiste en termes de performances, on voit qu'il est possible d'atteindre 8 à 10 GOPs/mm^2. La même figure présente le cas où des processeurs 24 bits dénués de mémoire de travail traitent des pixels de 8 bits. Dans ce cas, la capacité de calcul est très nettement augmentée puisqu'il est possible de dépasser 20 GOPs/mm^2 avec la même architecture. En supposant que des processeurs *SplitWay* 16 bits sont utilisés, cette capacité de calcul peut encore être significativement augmentée puisque nous avons vu que la surface du processeur est près de trois fois moindre (ce qui permet d'en intégrer trois fois plus). Ces

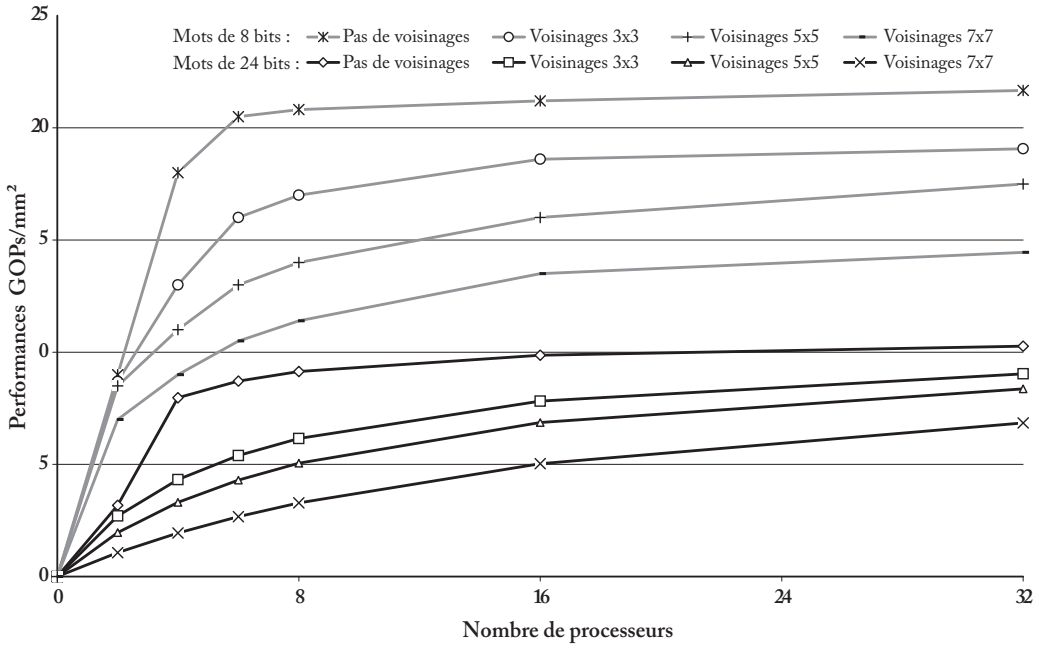


FIG. 5.8: CAPACITÉ DE CALCUL À 250 MHZ PAR UNITÉ DE SURFACE EN EXPRIMÉE GOPS/MM^2 EN FONCTION DU NOMBRE DE PROCESSEURS ET DE LA TAILLE DES VOISINAGES UTILISÉS – PROCESSEURS *SplitWay* 24 BITS ASSOCIÉS À DES MÉMOIRES DE TRAVAIL DE 256 MOTS DE 24 BITS.

résultats nous permettent de voir que les tuiles de calcul nécessitent entre 8 et 16 processeurs par tuile pour présenter une capacité de calcul élevée, ce qui confirme les résultats précédemment énoncés dans ce chapitre.

5.2.4 Capacité de calcul par unité de consommation

Le processeur *SplitWay* La consommation électrique du processeur *SplitWay* est de 0,84 mW à 133 MHz et de 1,81 mW à 200 MHz. Sa capacité de calcul par unité de consommation est donc respectivement de 316 MOPs/mW à 133 MHz, et 220 MOPs/mW à 200 MHz. L'étude en consommation n'a pas été réalisée sur une instance du processeur 16 bits, et ces chiffres excluent la mémoire de travail.

L'architecture *eISP* Les résultats de l'étude en consommation sont exploités pour déterminer la capacité de calcul par mW de l'architecture *eISP*. Ces résultats sont obtenus pour une fréquence de fonctionnement de 200 MHz afin de présenter des résultats conformes au fonctionnement nominal de l'architecture. La Figure 5.9 présente la capacité de calcul en MOPs/mW en fonction du nombre de processeurs *SplitWay* 24 bits et de la taille des voisinages gérés. La taille des mots de données est de 24 bits ce qui nous place, comme précédemment, dans un cas pessimiste, bien que nous ayons pu voir que la taille des données a un impact limité sur la consommation électrique. Cette figure montre qu'il est possible d'atteindre plus de 50 MOPs/mW dès que 8 processeurs sont utilisés à 200 MHz. Nous avons vu que la réduction de la fréquence de fonctionnement permet d'augmenter cette capacité de calcul par unité de consommation. Toutefois la capacité de calcul globale de l'architecture s'en trouve réduite.

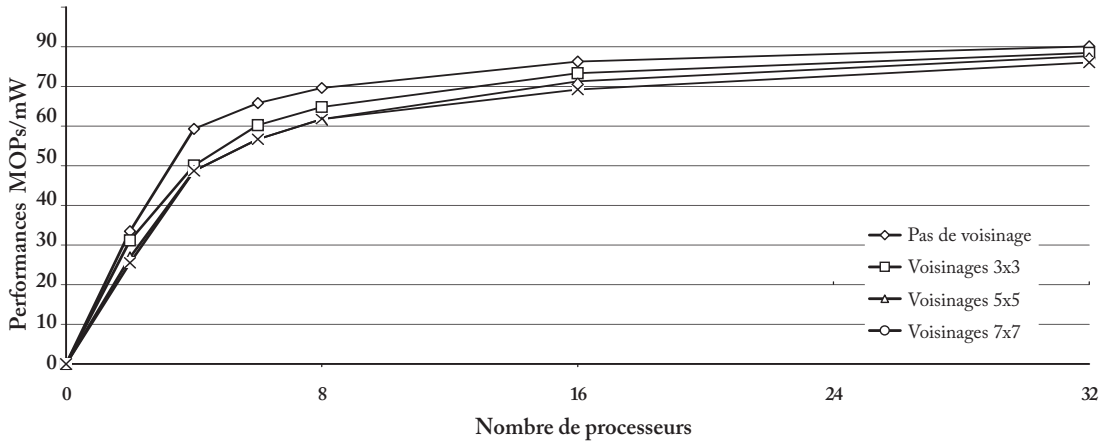


FIG. 5.9: CAPACITÉ DE CALCUL À 200 MHZ PAR UNITÉ DE SURFACE EN EXPRIMÉE MOPS/MW EN FONCTION DU NOMBRE DE PROCESSEURS ET DE LA TAILLE DES VOISINAGES UTILISÉS – PROCESSEUR *SplitWay* 24 BITS ASSOCIÉ À DES MÉMOIRES DE TRAVAIL DE 256 MOTS DE 24 BITS.

Conclusion de la caractérisation des performances de l'architecture

Après avoir vu que la surface silicium était principalement occupée par le gestionnaire de voisinages et les processeurs de calcul, cette section s'est proposée d'estimer la capacité de calcul de l'architecture *eISP*. Cette estimation est d'abord réalisée en tenant compte de l'ensemble des éléments qui composent une tuile de calcul, c'est à dire en estimant les ressources des processeurs *SplitWay*, mais aussi des gestionnaires de voisinages et des autres composants. Afin de tenir compte de la contrainte en surface et en consommation nous avons introduit la notion de capacité de calcul par unité de surface et par unité de consommation. Alors que le processeur *SplitWay* peut atteindre une capacité de calcul de plus de 300 MOPS/mW et 40 GOPs/mm^2, l'architecture *eISP* qui lui associe différents éléments de mémorisation et de calcul peut atteindre jusqu'à 20 GOPs/mm^2 pour plus 50 MOPs/mW de capacité de calcul effective, soit une consommation de 400 mW par mm^2 au pire cas.

Enfin l'étude en surface et en consommation permet de démontrer que l'architecture *eISP* est apte à répondre aux contraintes initialement fixées tant en surface qu'en consommation puisque de 5 à 20 GOPs/mm^2 sont atteints pour bien moins de 500 mW, en accord avec le cahier des charges. La structure modulaire de l'architecture, qui lui confère sa flexibilité, rend complexe les choix à réaliser, tant en termes de nombre de tuiles de calcul, que du nombre de processeurs *SplitWay* à intégrer par tuile, ou encore la largeur de leur chemin de données. Aussi est-il nécessaire de déterminer le jeu de tuiles le plus à même d'être exploité efficacement en fonction des classes d'algorithmes utilisés. Il est ensuite nécessaire de répartir les différents traitements sur ces tuiles afin de proposer une adéquation algorithme architecture optimale, qui peut être hétérogène.

5.3 Création d'instances de l'architecture adaptée au traitement HD 1080p

La première partie de ce chapitre a montré que les composants de l'architecture *eISP* peuvent être générés en fonction de différents paramètres, qui ont une influence sur la capacité de calcul et sur la flexibilité globale de l'architecture, mais aussi sur la surface silicium et la consommation électrique. Parmi ces paramètres, nous retiendrons d'abord le nombre de tuiles de calculs sur lesquelles un ou plusieurs traitements peuvent être portés et dont le programmeur peut en paramétrer l'enchaînement. Le nombre de processeurs *SplitWay* que ces tuiles de calcul intègrent est un facteur prépondérant de la capacité de calcul, mais aussi de la consommation élec-

trique du système ainsi que sa surface silicium. Il en va de même pour la mémoire de travail associée à chacun d'eux. Enfin la surface des tuiles de calcul dépend fortement de la taille maximale des masques de voisinages rendus accessibles aux processeurs mais aussi de la taille des mots de données manipulés. Un compromis en fonction des contraintes et des algorithmes doit donc être déterminé.

Pour cela, il est nécessaire d'identifier les propriétés des algorithmes portés sur l'architecture *eISP* afin d'en proposer une instance adaptée. Quelques éléments de la programmation de l'architecture *eISP* sont présentés au travers de l'exemple simple qu'est l'algorithme du démosaïquage bilinéaire. Une instance, appelée *Vid'eISP.1*, capable de supporter en temps réel des traitements vidéo HD 1080p. Elle est construite à partir des algorithmes des chaînes de référence vus dans le premier chapitre est ensuite présentée en 5.3.2, par application une démarche d'Adéquation-Algorithme-Architecture. Enfin, deux exemples de chaînes de traitements contraintes à un millimètre carré de surface silicium en technologie TSMC 65 nm sont présentées en 5.3.3. L'une privilégie la capacité de calcul et l'autre la flexibilité.

5.3.1 Programmation de l'architecture eISP

Chacun des algorithmes de la chaîne d'acquisition et d'amélioration d'image peut être porté sur une ou plusieurs tuiles de calcul programmables. Dans un premier temps, nous rappelons les particularités de la programmation SIMD et Multi-SIMD de l'architecture *eISP*. Dans un second temps, la programmation d'une tuile de calcul de l'architecture *eISP* est illustrée au travers de l'algorithme de démosaïquage bilinéaire. Enfin nous généraliserons en abordant les aspects liés à la programmation d'une architecture composée de plusieurs tuiles de calcul.

5.3.1.1 Particularités de la programmation de l'architecture eISP

Particularités du mode SIMD Lorsque les tuiles de calcul de l'architecture *eISP* sont configurées pour fonctionner en mode SIMD, Le même programme est déroulé pour l'ensemble des processeurs, chacun étant alimenté en données différentes par l'architecture. Le programmeur n'a donc qu'un seul programme à écrire, qui reste identique quel que soit le nombre de processeurs utilisés. Comme l'ensemble des processeurs exécutent simultanément les même instructions, ils ne peuvent réaliser de sauts individuels, l'unité de contrôle ne pouvant transmettre des opérations différentes relatives à ces sauts à chacun des processeurs. Lorsqu'un saut est exécuté, il concerne l'ensemble des processeurs. Afin de pouvoir exécuter des portions de code différentes en fonction des valeurs que peuvent prendre les données à traiter, ou encore en fonction de résultats intermédiaires, des opérations conditionnelles, définies au niveau processeur peuvent être employées. Des drapeaux peuvent être levés en fonction de la valeur des résultats et conditionnent ensuite de une ou plusieurs opérations. Soit l'opération concernée est exécutée, soit elle est remplacée par une opération vide.

Cette méthode assure l'exécution de prédicats `IF/ELSE` par l'ensemble des processeurs, et dans ce cas, le temps d'exécution correspond au temps de l'exécution des deux segments. La Figure 5.10 présente le cas d'un prédicat `IF` porté sur l'architecture *eISP*, où on voit que le processeur *SplitWay* VLIW deux voies est mis à profit pour réduire significativement le nombre d'instructions à exécuter, et donc le temps d'exécution. Afin de supporter plusieurs niveaux d'imbrication de `IF`, les valeurs de plusieurs drapeaux peuvent être mémorisées mais peuvent aussi être gérées comme des données par les opérateurs, ce qui permet de les sauvegarder, de les combiner ou encore de les transmettre à un processeur voisin.

Particularités du mode Multi-SIMD Par exemple, lorsque les images brutes qui alternent les couleurs primaires selon une matrice 2×2 doivent être traitées, quatre segments de codes sont généralement exécutés,

Implémentation « usuelle » **Prédicat `IF` « usuel »**	**Implémentation de type *SplitWay*** **Voie 1**	**Voie 2**
1: **if** TEST **then**	1: TEST F0	1:
2: SEGMENT A	2: F0 A OP1	2: NF0 B OP1
3: 4 opérations	3: F0 A OP2	3: NF0 B OP2
4: **else**	4: F0 A OP3	4: NF0 B OP3
5: SEGMENT B	5: F0 A OP4	5:
6: 3 opérations		
7: **end if**		

FIG. 5.10: EXEMPLE DE PORTAGE DE PRÉDICAT `IF` À L'AIDE D'INSTRUCTIONS CONDITIONNELLES, ICI LE DRAPEAU UTILISÉ EST LE DRAPEAU `F0`, LES PRÉFIXES `F0` ET `NF0`P PERMETTENT RESPECTIVEMENT D'EXÉCUTER L'OPÉRATION SUR LA VALEUR À VRAI ET À FAUX DU DRAPEAU.

un correspondant à chaque couleur. La position de chacun des pixels dans la matrice du filtre de couleur, est automatiquement associée à leur valeur, et est transmise à l'unité de contrôle. Cette opération est réalisée en temps masqué par le gestionnaire de voisinages qui possède déjà tous les éléments nécessaires à celle-ci. L'unité de contrôle est donc en mesure de transmettre aux processeurs les segments de codes correspondant aux pixels qu'ils ont à traiter. Cette méthode permet de supporter indifféremment tous types de filtres de couleur puisque ils sont logiciels.

Utilisation de la mémoire de travail Une mémoire de travail dont la taille – par exemple 256 mots de 24 bits – est définie à la création de l'architecture et est associée à chacun des processeurs *SplitWay*. Outre la possibilité de sauvegarder des valeurs, cette mémoire est utilisée pour intégrer des LUTs aux processeurs. Par exemple, une mémoire de travail de 256 mots de 24 bits permet l'utilisation de 3 LUTs de 8 bits chacune. Le programmeur peut ainsi utiliser de manière transparente des opérations complexes. L'initialisation des valeurs de ces LUTs est effectuée lors de la configuration de la tuile de calcul, tandis que la chaîne d'outils de l'architecture *eISP* assure au programmeur un accès transparent aux opérations qu'il peut prédéfinir. Ces opérations sont de la forme $y = f(x)$, et peuvent correspondre à des divisions, des fonctions exponentielles, ou des filtres plus complexes comme une correction gamma. Les LUTs peuvent être reprogrammées pour chaque traitement, ce qui rend flexible le choix des opérateurs supplémentaires qu'il est possible d'intégrer à l'architecture.

Accès aux données Les processeurs *SplitWay* ont un jeu d'instructions orthogonal qui leur permet d'utiliser des données issues de sources différentes comme opérandes. De cette manière, les données du gestionnaire de voisinages sont donc directement utilisables. Ces sources sont multiples, comme le plan mémoire externe, ou des registres partagés par les processeurs voisins ou de tout autre composant de la tuile de calcul si cela est nécessaire. De plus, un gestionnaire de métadonnées permet de sélectionner la partie du mot sur lequel réaliser les traitements.

Exportation des résultats et communication Les résultats des calculs sont automatiquement sérialisés pour constituer un flux de pixels. Les valeurs sérialisées sont issues de la file de registres de chacun des processeurs. Dans sa configuration standard, il s'agit des registres `R1`, `R2` et `R3`. Ces registres restent manipulables pendant toute la durée d'exécution d'un segment de code. En effet, leurs valeurs sont exportées à la fin du temps

imparti pour le traitement du pixel, pendant le lancement du code correspondant au pixel suivant à exécuter. Ce flux est ensuite transmis aux autres tuiles de calcul par l'intermédiaire du module de communication que le programmeur peut paramétrer en fonction de ses besoins.

Le programmeur peut choisir quelles valeurs exporter et dans quel format grâce à la définition du format des métadonnées. Par exemple, seuls les 12 premiers bits du registre `R2` peuvent être envoyés à une autre tuile, ou encore la concaténation des 8 premiers bits de chacun des trois registres, ce qui peut correspondre à trois composantes. Le programmeur peut aussi choisir d'associer la valeur d'un résultat intermédiaire, qui peut être le résultat d'un calcul ou une adresse à la valeur d'un pixel. Les différentes configurations ainsi possibles permettent de supporter les nombreuses classes d'algorithmes de traitement d'image tout en prévoyant des cas d'utilisation qui assurent à l'architecture la flexibilité nécessaire pour que le programmeur puissent implémenter différents types d'algorithmes.

Fréquence de fonctionnement Nous avons vu que plusieurs arbres d'horloges peuvent cadencer les différentes tuiles de calcul. Cette caractéristique permet d'ajuster au mieux la capacité de calcul aux ressources que nécessitent les traitements portés sur les tuiles de calcul tout en limitant la consommation électrique. Comme nous l'avons vu précédemment, la communication entre les tuiles de calcul est assurée par un bus TDMA. Mais ce bus a aussi pour fonction de synchroniser les tuiles entre elles en transportant le flux de pixel à traiter. La fréquence de fonctionnement optimale de chaque tuile étant automatiquement déterminée à la compilation du programme.

Configuration de la tuile de calcul Lorsqu'un algorithme est porté sur une tuile de calcul, celle-ci est configurée en conséquence. Le programme est effectivement écrit en mémoire programme selon le mode qui est choisi par le programmeur (SIMD ou Multi-SIMD). La taille des voisinages à utiliser est transmise au gestionnaire de voisinages tandis que le format des données de sortie est transmis aux modules d'entrée-sorties. Ce dernier reçoit aussi le numéro du slot temporel et du canal sur lequel il doit lire les données d'entrées, ainsi que celui sur lequel les résultats doivent être exportés. De plus, les mémoires de travail internes aux processeurs sont initialisées, puis la fréquence de fonctionnement optimale des processeurs est configurée en fonction de la taille des segments de code à exécuter avec comme stratégie le choix de la fréquence de fonctionnement la plus basse possible qui assure le fonctionnement temps réel. Les formats choisis pour les métadonnées sont écrits dans les registres de masques pendant cette étape ainsi que les paramètres d'éventuelles extensions à la tuile de calcul. La gestion de l'ensemble de ces paramètres est assurée par l'unité de contrôle qui les transmet aux différents composants de sa tuile de calcul à partir des informations fournies par l'outil de de configuration de chacune des tuiles.

5.3.1.2 Exemple de programmation d'une tuile de calcul

Afin d'illustrer les différents aspects de la programmation d'une tuile de calcul, l'exemple de l'algorithme de démosaïquage bilinéaire est présenté. Au cours de cet exemple, nous considérerons la tuile de calcul présentée précédemment au cours de l'étude en consommation 5.2. Cette tuile de calcul est composée de 6 processeurs *SplitWay* et une mémoire de travail de 256 mots de 24 bits est associée à chacun d'eux. De plus, ils ont accès à un voisinage 3×3 du pixel à traiter.

Cet algorithme doit convertir une image brute, généralement acquise par un capteur recouvert du filtre de Bayer en une image à pleine résolution de trois composantes couleurs. Par conséquent, le flux d'entrée correspond à une image brute, tandis que le flux de sortie est composé de trois composantes, rouge, verte et bleue.

```
1:  if i%2 == 0 then
2:     if j%2 == 0  then
3:        //Pixel Vert_Rouge
4:        R(i,j) = Px(i,j)
5:        G(i,j) = (Px(i-1,j) + Px(i+1,j) + Px(i,j-1) + Px(i,j+1))/4
6:        B(i,j) = (Px(i-1,j-1) + Px(i+1,j-1) + Px(i-1,j+1) + Px(i+1,j-1) )/4
7:     else if j%2== 1 then
8:        //Pixel Rouge
9:        R(i,j) = (Px(i-1,j) + Px(i+1,j))/2
10:       G(i,j) = Px(i,j)
11:       B(i,j) = (Px(i,j+1) + Px(i,j-1))/2
12:    end if
13: else if i%2 == 1 then
14:    if j%2 == 0 then
15:       //Pixel Bleu
16:       R(i,j) = (Px(i-1,j) + Px(i+1,j) + Px(i,j-1) + Px(i,j+1))/4
17:       G(i,j) = (Px(i-1,j-1) + Px(i+1,j-1) + Px(i-1,j+1) + Px(i+1,j-1) )/4
18:       B(i,j) = Px(i,j)
19:    else if j%2==1 then
20:       //Pixel Vert_Bleu
21:       R(i,j) = (Px(i,j+1) + Px(i,j-1))/2
22:       G(i,j) = Px(i,j)
23:       B(i,j) = (Px(i-1,j) + Px(i+1,j))/2
24:    end if
25: end if
```

FIG. 5.11: CŒUR DE BOUCLE DE L'ALGORITHME DE DÉMOSAÏQUAGE BILINÉAIRE. CETTE IMPLÉMENTATION FAIT APPARAÎTRE LES DIFFÉRENTES ÉQUATIONS À APPLIQUER EN FONCTION DE LA COULEUR DU PIXEL À TRAITER.

Ce flux d'entrée correspond à 52 MPixels par seconde à partir desquels 3 flux de 52 MPixels par seconde sont générés.

L'algorithme présenté en Figure 5.11 illustre les équations mises en œuvre dans le cœur de boucle parcourant l'image pixel après pixel. On voit sur cet exemple que le programme à exécuter est différent en fonction de la couleur du pixel à traiter. Le mode de fonctionnement Multi-SIMD est donc approprié pour ce type d'algorithme. L'étude de cet algorithme montre qu'un accès 3×3 au voisinage est suffisant. De plus, la mémoire de travail restera inutilisée puisqu'aucun opérateur complexe ne nécessite d'être implémenté sous forme de LUTs tandis qu'aucune donnée de taille conséquente ne doit être sauvegardée. Enfin les résultats correspondant aux trois composantes rouge verte, et bleue de chaque pixel sont écrits respectivement dans les registres `R1`, `R2` et `R3`, qui seront exportés vers une autre tuile de calcul par le module communication.

Algorithme bilinéaire porté sur l'architecture *eISP*. Le code du programme tel qu'il est décrit par le programmeur est présenté en Figure 5.12. Cette figure présente les quatre segments de codes correspondant aux quatre cas rencontrés dans un filtre de couleur 2×2. Notons que si la disposition du filtre est modifiée, il suffit de « déplacer » les segments de code correspondants. Cet exemple permet de voir que seules les équations du

cœur de boucle sont écrites. Ainsi le segment le plus long de démosaïquage bilinéaire nécessite seulement 6 instructions, soit 6 cycles processeurs par pixel, il s'agit des segments appelés `PixelCode0` et `PixelCode3`. En effet aucun calcul d'adresse n'est effectué par le programme puisque les voisinages associés au pixel à traiter sont directement accessibles.

Les processeurs *SplitWay* intègrent deux additionneurs/soustracteurs ainsi que deux unités d'affectations rendant possible l'exécution simultanée de ces opérations sur les deux voies, comme c'est par exemple le cas à la ligne 8 du programme détaillé en Figure 5.12. La tuile de calcul possède 6 processeurs *SplitWay*, il est donc possible de déterminer la fréquence de fonctionnement la plus faible assurant un taux d'utilisation maximal[1]. La Table 5.5 présente le taux d'utilisation des processeurs de cette instance de la tuile de calcul en fonction de leurs fréquences de fonctionnement. Cette table montre que même à des fréquences de fonctionnement faibles de 133 MHz, la tuile de calcul intégrant le démosaïquage bilinéaire est nettement sous utilisée.

Cet exercice montre qu'il est envisageable de porter des traitements supplémentaires, ou plus complexes, sur une telle tuile de calcul. Par ailleurs, augmenter la cadence du flux vidéo rend un telle tuile utilisable pour des applications de vidéo rapide, puisque plus de 200 MPixels par seconde peuvent ainsi être démosaïqués si la tuile de calcul est cadencée à 250 MHz, ce qui correspond à des flux vidéo de 650 images par seconde à 640×480. Une tuile de calcul comportant 32 processeurs à 250 MHz utilisant le même algorithme peut démosaïquer des flux vidéo de l'ordre de tera-pixel par seconde pour une surface silicium de l'ordre de 1 mm^2 et une capacité de calcul effective de 16 GOPs.

TAB. 5.5: TAUX D'UTILISATION DES PROCESSEURS EN FONCTION DE LEUR FRÉQUENCE DE FONCTIONNEMENT.

Fréquence en MHz	**75**	**80**	**90**	**100**	**120**	**133**	**200**	**250**
Cycles disponibles par pixel	6	7	8	9	11	13	21	26
Taux d'utilisation	100 %	85 %	75 %	66 %	54 %	46 %	28 %	23 %

5.3.2 Création d'une instance générique pour le traitement vidéo HD 1080p

Cette partie aborde la création d'une instance générique de l'architecture *eISP* adapté au traitement vidéo HD 1080p que nous appellerons *Vid'eISP.1*, et dont la surface silicium doit être inférieure à 3 millimètres carrés pour moins de 500 mW de consommation électrique. Pour cela, les caractéristiques de chaque algorithme de la tuile de calcul sont extraites (taille de voisinages, utilisation de la mémoire de travail etc.) comme nous l'avons fait pour l'algorithme bilinéaire. La Figure 3.3 de la page 49 nous rappelle les algorithmes en jeu dans ces chaînes de traitements ainsi que les capacités de calcul requises. La première chaîne de référence requiert 17,59 GOPs de capacité de calcul totale tandis que la seconde en requiert 70,89 GOPs et la troisième 20,25 GOPs. Les algorithmes sont ensuite portés sur cette instance *Vid'eISP.1*.

5.3.2.1 Ressources spécifiques au portage d'algorithmes

Les paramètres associés à chacun des algorithmes sont obtenus par leur portage sur l'architecture. Ces paramètres ayant un impact sur le choix de la tuile de calcul par rapport à ses ressources, sont :

– le nombre d'instructions (et donc de cycles) par pixel nécessaire au traitement ;

– le nombre d'instructions à exécuter en inter-trame ;

1. Le taux d'utilisation est obtenu par comparaison du nombre de cycles utilisés pour le calcul au le nombre de cycle totaux disponibles durant le temps de traitement imparti.

– le format des pixels d'entrée et de sortie (images brutes 12 bits, RVB 8 bits etc.) ;

– la taille des masques de voisinages ;

– la taille des mémoires de travail nécessaires (sauvegardes de résultats, opérateurs implémentés par des LUTs ;

– le mode de fonctionnement (SIMD ou Multi-SIMD).

Les tableaux 5.6 et 5.7 présentent les propriétés des deux premières chaînes de référence. La troisième n'est pas présentée puisqu'elle reprend les algorithmes des deux autres. Notons que le filtre médian peut efficacement être remplacé par un débruitage par filtre non linéaire, il consiste à remplacer les valeurs des pixels trop éloignés de la valeur moyenne du voisinage du pixel considéré par cette moyenne. De même, afin de limiter l'espace mémoire nécessaire à la normalisation d'histogramme, il est préférable de compresser la dynamique des valeurs des pixels de 12 bits à 8 bits.

Voie 1	Voie 2
1: .PixelCode0	1:
2: LDv R1 V[0, 0]	2: LDv R2 V[-1, 0]
3: ADDv R2 R2 V[-1, 0]	3: LDv R3 V[-1,-1]
4: ADDv R2 R2 V[+1, 0]	4: ADDc R3 R3 V[-1,+1]
5: ADDv R2 R2 V[0,+1]	5: ADDv R3 R3 V[+1, 0]
6: SHIFTLc R2 R2 2	6: ADDc R3 R3 V8
7: SHIFTLc R3 R3 2	7:
8: .PixelCode1	8:
9: LDv R1 V[0,-1]	9: LDv R3 V[-1, 0]
10: ADDv R3 R3 V[+1, 0]	10: ADDv R1 R1 V[+1,+1]
11: LDv R2 V[0, 0]	11: SHIFTLc R1 R1 1
12: SHIFTLc R3 R3 1	12:
13: .PixelCode2	13:
14: LDv R1 V[0, 0]	14: LDv R3 V[-1, 0]
15: ADDv R3 R3 V[-1, 0]	15: LDv R2 V[-1,-1]
16: ADDv R3 R3 V[+1, 0]	16: ADDc R2 R2 V[-1,+1]
17: ADDv R3 R3 V[+1,+1]	17: ADDv R2 R2 V[+1, 0]
18: SHIFTLc R3 R3 2	18: ADDc R2 R2 V[+1,+1]
19: SHIFTLc R2 R2 2	19:
20: .PixelCode3	20:
21: LDv R1 V[0,-1]	21: LDv R2 V[-1, 0]
22: ADDv R2 R2 V[-1, 0]	22: ADDv R1 R1 V[+1,+1]
23: LDv R3 V[0, 0]	23: SHIFTLc R1 R1 1
24: SHIFTLc R2 R2 1	24:

FIG. 5.12: LE PROGRAMME DE L'ALGORITHME DE DÉMOSAÏQUAGE BILINÉAIRE TEL QU'IL EST PORTÉ SUR UNE TUILE DE L'ARCHITECTURE *eISP*.

TAB. 5.6: PROPRIÉTÉS DES ALGORITHMES DE LA PREMIÈRE CHAÎNE DE RÉFÉRENCE PORTÉS SUR L'ARCHITECTURE *eISP*.

Num.	Algorithme	Instructions par pixel	Instructions inter-trames	Taille des voisinages	Taille des mem. trav.	Format d'entrée	Format de sortie
1	Débruitage par filtre médian	88	0	5×5	0	Brute 8 bits	Brute 8 bits
$1^{alternative}$	Débruitage par filtre non linéaire	13	0	5×5	0	Brute 8 bits	Brute 8 bits
2	Débruitage par flou gaussien	15	0	5×5	0	Brute 8 bits	Brute 8 bits
3	Normalisation d'histogramme	8	$NbProc \times 2^8$	1×1	2^8	Brute 12 bits	Brute 8 bits
4	Estimation et correction de la balance des blancs	3	3×2^8	1×1	0	Brute 8 bits	Brute 8 bits
5	Démosaïquage par interpolation bilinéaire	6	0	3×3	0	Brute 8 bits	RVB 8 bits
6	Accentuation des contours	3×16	0	3×3	0	RVB 8 bits	RVB 8 bits

TAB. 5.7: PROPRIÉTÉS DES ALGORITHMES DE LA SECONDE CHAÎNE DE RÉFÉRENCE PORTÉS SUR L'ARCHITECTURE *eISP*.

Num.	Algorithme	Instructions par pixel	Instructions inter-trames	Taille des voisinages	Taille des mem. trav.	Format d'entrée	Format de sortie
1	Débruitage par filtre médian	88	0	5×5	0	Brute 8 bits	Brute 8 bits
$1^{alternative}$	Débruitage par filtre non linéaire	13	0	5×5	0	Brute 8 bits	Brute 8 bits
2	Filtre bilatéral	34	0	5×5	$2^8 \times 3$	Brute 8 bits	Brute 8 bits
3	Normalisation d'histogramme	8	$NbProc \times 2^8$	1×1	2^8	Brute 8 bits	Brute 8 bits
4	Correction gamma locale adaptative	21	0	5×5	2^8	Brute 8 bits	Brute 8 bits
5	Estimation et correction de la balance des blancs	3	3×2^8	1×1	0	Brute 8 bits	Brute 8 bits
6	Démosaïquage par interpolation de Hamilton & Adams (1)	43	0	5×5	0	Brute 8 bits	Composé 16 bits
6^{suite}	Démosaïquage par interpolation de Hamilton & Adams (2)	35	0	5×5	0	Composée 16 bits	RVB 8 bits
7	Accentuation des contours	3×16	0	3×3	0	RVB 8 bits	RVB 8 bits

5.3.2.2 Problématique de la fusion d'algorithmes

Afin de limiter le nombre de tuiles de calcul et ainsi éviter de dupliquer l'infrastructure d'accès aux données et de contrôle des processeurs, nous étudions la possibilité de regrouper plusieurs algorithmes. Comme chaque algorithme dépend des résultats du traitement précédent, il s'avère parfois très coûteux en termes d'instructions de les regrouper sur une même tuile.

Prenons par exemple une correction non linéaire, comme un seuillage, suivi d'une convolution 3×3. Si les traitements sont réalisés sur la même tuile de calcul, il est nécessaire de lui appliquer l'algorithme de seuillage. Du point de vue des processeurs, les gestionnaires de voisinages sont en lecture seule, il est donc impossible de précalculer la valeur, la sauvegarder et la réutiliser ensuite autrement que par l'utilisation d'une tuile de calcul précédente. Notons que si l'ordre d'exécution des algorithmes de cet exemple est inversé – la fonction réalisée est alors différente de la précédente – c'est à dire, que le seuillage est effectué après la convolution, alors il est possible de les intégrer sans surcoût dans la même tuile de calcul, ceci est dû au fait que le seuillage ne nécessite pas l'accès aux valeurs du voisinage du pixel à traiter.

Nous avons spécifiquement étudié cette problématique au niveau de l'ensemble des ressources en jeu (éléments de mémorisation, etc.), ce qui a conduit, en première approche, à proposer un outil d'adéquation-algorithme-architecture appelé *eISP-OptAC* (pour *Area and Consumption Optimization*) permettant une répartition efficace des algorithmes en fonction des contraintes en surface et consommation [Ngathe-Simo 2009] et des caractéristiques des algorithmes qui sont donnés. Il apparaît qu'il n'est pas optimal et particulièrement peu flexible, de proposer une seule tuile de calcul composée d'un grand nombre de processeurs, par rapport à une solution où plusieurs tuiles de calcul permettent d'enchaîner les traitements selon les choix du programmeur. Ainsi, chaque algorithme de la chaîne de traitement est porté sur une tuile, excepté la correction de la balance des blancs qui pourra partager celle où est porté le débruitage du bruit blanc gaussien, cinq tuiles de calcul sont nécessaires pour la première chaîne de traitement, et sept tuiles de calcul sont nécessaires pour la seconde. De même, il s'avère judicieux de scinder les traitements en plusieurs parties.

5.3.2.3 Génération des tuiles de calcul

À partir des propriétés des algorithmes et de leur répartition, nous utilisons l'outil *geneSP* qui génère les tuiles de calcul adéquates en fonction de leurs paramètres. Tous les résultats présentés correspondent à l'utilisation de technologie TSMC 65 nm. Nous proposons une structure hétérogène, composée de plusieurs groupes de tuiles de calcul. Toutefois afin de maintenir un maximum de flexibilité, toutes les tuiles intègrent un gestionnaire de voisinages 5×5. Nous choisissons de conserver des instances 24 bits du processeur de calcul pour permettre une évolution ultérieure des algorithmes ou des formats de données, bien que pour la plupart des algorithmes qui nous intéressent ici, des instances 16 bits se révèlent suffisantes.

La Table 5.8 présente les trois types de tuiles de calcul et leur propriétés qui constituent *Vid'eISP.1*, sur lesquels les algorithmes des chaînes de traitement peuvent être portés. Les tuiles de type « A » présentent des ressources de mémorisation importantes adaptées aux traitements utilisant des LUTs pour opérateurs. Les tuiles « B » sont capables de traiter des mots de données de 24 bits et possédant des capacités de calcul moyennes, enfin la tuile de calcul de type « C » intègre peu de ressources mémoire mais a une capacité de calcul élevée de près de 200 opérations par cycle à 250MHz. Ces trois types de tuiles présentent toutes une surface de l'ordre d'un tiers de millimètre carré. Cette variété de tuiles calcul générées montre la flexibilité du modèle de l'architecture *eISP*. Ces tuiles de calcul sont interconnectées entre elles par trois canaux TDMA de 12 bits et un canal de 24 bits, ainsi toutes les tuiles de calcul peuvent communiquer entre elles.

TAB. 5.8: INSTANCE SEMI-HÉTÉROGÈNE D'EISP, CAPABLE DE DÉLIVRER 40 GOPS À 250 MHZ POUR 295 MW DE CONSOMMATION ÉLECTRIQUE SUR 2,17 MM^2 DE SURFACE SILICIUM .

Num.	Nombre de tuiles	Nombre de processeurs	Taille de la mémoire	Taille max des voisinages	Taille des mots de données	Surface silicium	Conso. élec.
A	3×	8	256×24 *bits*	5×5	12 bits	3× 0,30mm^2	30 mW
B	3×	12	0	5×5	24 bits	3× 0,30mm^2	45 mW
C	1×	20	0	5×5	8 bits	0,34 mm^2	70 mW
Total	7	80	18 kbits	NA	NA	2,17mm^2	295 mW

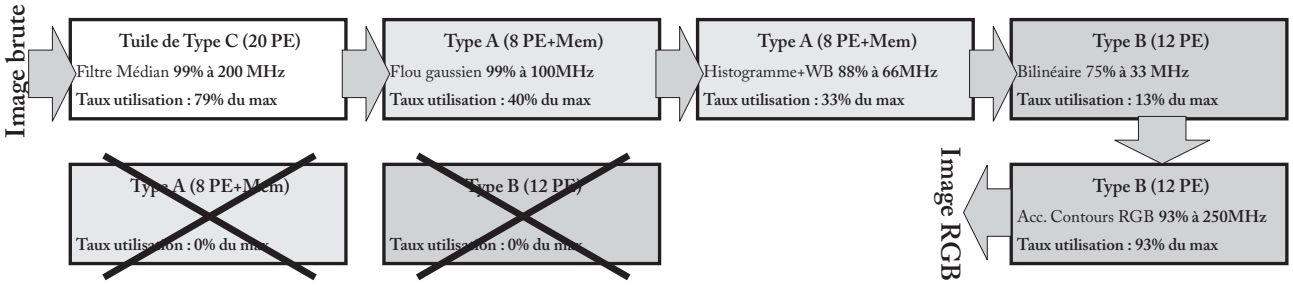


FIG. 5.13: PORTAGE DE LA PREMIÈRE CHAÎNE DE RÉFÉRENCE SUR *Vid'eISP.1*, LE TAUX D'UTILISATION DE LA TUILE DE CALCUL EST PRÉSENTÉ POUR LA FRÉQUENCE DE FONCTIONNEMENT CHOISIE, ET PAR RAPPORT À SA CAPACITÉ DE CALCUL À 250 MHZ, DEUX TUILES SONT INUTILISÉES, (*WB* SIGNIFIE « BALANCE DES BLANCS »).

5.3.2.4 La répartition des algorithmes sur les tuiles de calcul

Comme toutes les tuiles de calcul peuvent communiquer entre elles, leur enchaînement n'est pas un critère de répartition. La répartition optimale de ces tuiles est un problème qui dépasse le cadre de ce manuscrit et qui relève d'outils logiciels d'Adéquation-Algorithme-Architecture. Nous proposons donc une répartition « manuelle »assistée. Les Figures 5.13, 5.14 et 5.15 représentent les répartitions des algorithmes de référence sur les tuiles de calcul de l'architecture *Vid'eISP.1* proposée. Leur taux d'utilisation pour une fréquence donnée et pour le traitement d'un flux vidéo HD 1080p (1920×1080 à 25 images par seconde) est présenté. La grande majorité des tuiles fonctionne à une fréquence très inférieure à la fréquence maximale prévue, y compris pour le cas de chaîne numéro deux qui se veut la plus calculatoire avec ses 70,89 GOPs. Cet exemple illustre pleinement l'utilité de l'utilisation d'une fréquence de fonctionnement adaptée à chaque tuile de calcul. Notons que le filtre médian implémenté ici utilise un algorithme de tri, son utilisation, ainsi que celle du filtre bilatéral permet de démontrer la capacité de l'architecture *Vid'eISP.1* à supporter des calculatoires complexes . Le lecteur peut se référer aux trois chaînes de reconstruction introduites en 1.3.3, et notamment aux Figures 1.11, 1.12 et 1.13 et leurs pages respectives 23, 24 et 25.

5.3.2.5 Résultats obtenus

L'architecture *Vid'eISP.1* obtenue est composée de sept tuiles de calcul, pour un total de 80 processeurs *SplitWay* 24 bits intégrant tous un multiplieur. 18 kbits de mémoire sont distribués sur 3 tuiles de 8 processeurs, et elle supporte des tailles de voisinage allant jusqu'à 5×5 pour des lignes de 2048 mots. La tuile de calcul la plus performante intègre 20 processeurs et peut assurer 196 opérations par pixel pour des vidéo HD 1080p, et 1250 opérations par pixel lorsque des vidéos au format VGA doivent être traitées. La capacité de calcul effective de *Vid'eISP.1* est de 40 GOPs soit 18,43 GOPs/mm^2 et 135 MOPs/mW, et sa capacité de calcul équivalente est

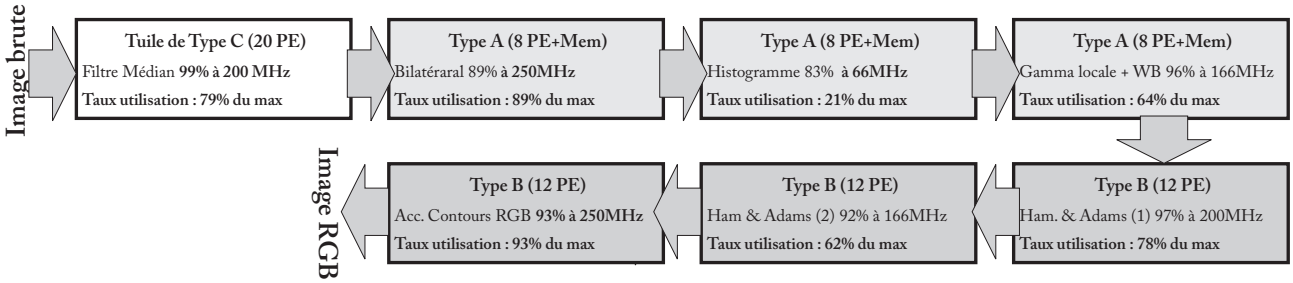


FIG. 5.14: PORTAGE DE LA SECONDE CHAÎNE DE RÉFÉRENCE SUR *Vid'eISP.1*, LE TAUX D'UTILISATION DE LA TUILE DE CALCUL EST PRÉSENTÉ POUR LA FRÉQUENCE DE FONCTIONNEMENT CHOISIE, ET PAR RAPPORT À SA CAPACITÉ DE CALCUL À 250 MHZ, TOUTES LES TUILES SONT UTILISÉES, MAIS UNE IMPORTANTE CAPACITÉ DE CALCUL RESTE DISPONIBLE (*WB* SIGNIFIE « BALANCE DES BLANCS »).

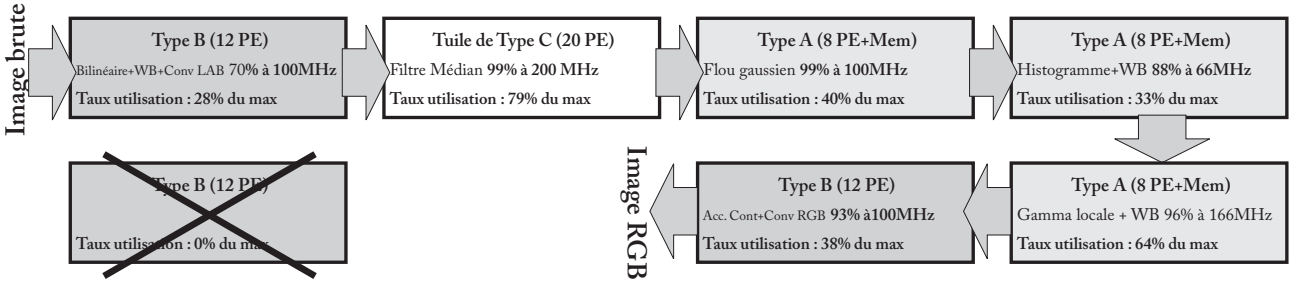


FIG. 5.15: PORTAGE DE LA TROISIÈME CHAÎNE DE RÉFÉRENCE SUR *Vid'eISP.1*, LE TAUX D'UTILISATION DE LA TUILE DE CALCUL EST PRÉSENTÉ POUR LA FRÉQUENCE DE FO NCTIONNEMENT CHOISIE, ET PAR RAPPORT À SA CAPACITÉ DE CALCUL À 250 MHZ, DEUX TUILES SONT INUTILISÉES (*WB* SIGNIFIE « BALANCE DES BLANCS »).

estimée entre 95 GOPs et 136 GOPs.

La consommation électrique de *Vid'eISP.1* est estimée à 295 mW pour une fréquence de fonctionnement nominale de 250 MHz avec une surface à 2,17 mm^2. Le mode Multi-SIMD simplifié est utilisé, trois tuiles sont en mesure de supporter des images codées sur 12 bits, trois tuiles sont prévues pour le traitement de données allant jusqu'à 24 bits, et enfin une tuile pour des mots de 8 bits. Ces résultats tiennent compte des modules de communication qui représentent 12 % de la surface silicium. Le bus TDMA est composé de 4 canaux indépendants de 12 bits, et 2 canaux supplémentaires de 24 bits, tous cadencés à une fréquence unique.

5.3.3 Exemple d'instances d'eISP d'un millimètre carré

Afin de démontrer la flexibilité des éléments de l'architecture *eISP*, cette sous-section présente deux cas d'utilisation pour lesquels la surface silicium est limitée à un millimètre carré environ, il s'agit là d'une contrainte forte. Dans le premier cas, la capacité de calcul est recherchée, mais sans réduire pour autant la flexibilité. Dans le second cas, c'est un niveau de flexibilité plus élevé qui est recherché notamment en intégrant suffisamment d'éléments de mémorisation aux tuiles de calcul. Dans les deux cas présentés, les tuiles de calcul sont destinées à supporter des flux vidéo HD 1080p.

TAB. 5.9: TABLEAU DE SYNTHÈSE PRÉSENTANT LES TUILES SÉLECTIONNÉES POUR UNE INSTANCE DE L'ARCHITECTURE *eISP* PRIVILÉGIANT LA CAPACITÉ DE CALCUL.

Nb	Nombre de processeurs	Taille des voisinages	Taille mots de données	Mém. de trav.	Surface	Consommation à 250 MHz	Capacité de calcul à 250 MHz
1×	22 de 24 bits	5×5	12 bits	0 kbit	0,45 mm^2	57 mW	11 GOPs
2×	22 de 16 bits	3×3	8 bits	0 kbit	0,19 mm^2	30 mW	11 GOPs
2×	22 de 16 bits	1×23	8 bits	0 kbit	0,10 mm^2	22 mW	11 GOPs
5	**110**			0 kbit	1,03 mm^2	159 mW	55 GOPs

5.3.3.1 Capacité de calcul privilégiée

Stratégie adoptée Dans ce premier cas, afin d'assurer un niveau de flexibilité permettant le portage de la plupart des chaînes de traitement, nous chercherons à intégrer au moins 4 tuiles de calcul à cette instance de l'architecture *eISP*. Elles sont organisées telles que deux d'entre elles puissent traiter des flux de pixel 12 bits avec des voisinages 5×5 et les deux autres traitent des pixels 8 bits avec des voisinages 3×3. Enfin des tuiles n'intégrant pas de gestionnaire de voisinages sont proposées au programmeur afin d'exécuter des calculs sur des voisinages à une dimension $M \times 1$. Pour une partie des tuiles de calcul, nous choisissons de réduire le chemin de données des processeurs *SplitWay* ainsi que la dynamique des opérateurs à 16 bits. Les processeurs sont moins volumineux ce qui nous permet d'en augmenter le nombre. Des processeurs 16 bits permettent de couvrir une large gamme de traitements. De plus, afin d'utiliser aux mieux la surface silicium, la mémoire de travail associée à ces processeurs est totalement supprimée. Cette version est comparable aux architectures de l'état de l'art.

La Table 5.9 présente les tuiles de calcul choisies pour constituer cette architecture de calcul. A surface équivalente, il est possible d'augmenter la capacité de calcul de chaque tuile en leur ajoutant des processeurs, au détriment d'une tuile. Ainsi, passer de 16 à 22 processeurs par tuile permet l'exécution d'algorithmes nécessitant plus de 200 opérations par cycle sur des flux vidéo HD 1080p. Pour obtenir la surface silicium adéquate, la suppression d'une des tuiles de calcul est nécessaire, ce qui réduit la flexibilité du système. Dans ce cas la capacité de calcul s'en trouve augmentée, et passe à 55 GOPs.

Bilan architectural Une architecture *eISP* dotée de telles tuiles de calcul présente une surface totale de 1,03 mm^2 pour une consommation électrique estimée à 175 mW dans un cas de fonctionnement pessimiste à 250 MHz. La capacité de calcul ainsi développée à cette fréquence est de 55 GOPs, ce qui confère à cette architecture une capacité de calcul par unité de surface de 54 GOPs/mm^2 et 345 MOPs/mW à 250 MHz en termes de capacité de calcul par unité de consommation. À 133 MHz, cette capacité de calcul atteint 0,6 GOPs/mW, mais à une telle fréquence la capacité de calcul totale chute à moins de 28 GOPs/mm^2.

Ces résultats tiennent uniquement compte des capacités de calcul des processeurs, nous avons vu précédemment en 5.2 que si les capacités de calcul des éléments dédiés (gestionnaire de voisinages, unité de contrôle etc.) étaient prises en compte, il était possible d'ajouter – dans le cas de traitement de flux de pixels cadencés à 52 MPx/s pour une fréquence processeur de 250 MHz –, 48 GOPs aux tuiles intégrant des gestionnaires de voisinages 5×5, 32 GOPs à celles intégrant des gestionnaires de voisinages 3×3 et et 0.9 GOPs supplémentaires aux autres, soit un total de 113 équivalent GOPs supplémentaires aux 55 GOPs dédiés au calcul pour l'ensemble de l'architecture considérée. La capacité de calcul équivalente, en tenant compte des ressources dédiées peut

TAB. 5.10: TABLEAU DE SYNTHÈSE PRÉSENTANT LES TUILES SÉLECTIONNÉES POUR UNE INSTANCE DE L'ARCHITECTURE *eISP* PRIVILÉGIANT LA FLEXIBILITÉ.

Nb	Nombre de processeurs	Taille des voisinages	Taille mots de données	Mém. de trav.	Surface	Consommation à 250 MHz	Capacité de calcul à 250 MHz
1×	8 de 24 bits	3×3	12 bits	256×24 bits	0,24 mm^2	26 mW	4 GOPs
1×	8 de 24 bits	3×3	24 bits	256×24 bits	0,30 mm^2	28 mW	4 GOPs
2×	16 de 16 bits	3×3	8 bits	256×16 bits	0,22 mm^2	32 mW	8 GOPs
1×	16 de 16 bits	1×11	8 bits	0	0,09 mm^2	17 mW	8 GOPs
5	**64**			**163 kbits**	**1,07 mm^2**	**135 mW**	**32 GOPs**

est donc être évaluée à 163 GOPs/mm^2 et 1,02 GOPs/mW. Notons que la réduction de la taille du chemin de données peu permettre d'augmenter la fréquence de fonctionnement des tuiles concernées.

La flexibilité par rapport à l'instance *Vid'eISP.1* est moindre. L'absence de mémoire de travail ne permet pas, par exemple l'exécution d'algorithmes basés sur des analyses d'histogramme ou l'utilisation de LUTs.

5.3.3.2 Flexibilité privilégiée

Stratégie adoptée Nous avons vu que la flexibilité globale de l'architecture dépend du nombre de tuiles de calcul disponibles, mais aussi des éléments de mémorisation. Qu'il s'agisse des gestionnaires de voisinages, ou bien des mémoires de travail, ils induisent une importante surface silicium. Afin de limiter la surface qui leur est dédiée, nous choisissons de supporter des mots de 8 bits sur l'ensemble des tuiles de calcul, excepté une qui est en mesure supporter 12 bits, et une autre des mots de 24 bits pour exécuter des algorithmes nécessitant l'accès aux trois composantes d'une image pour déterminer un résultat. De plus, la réduction du nombre de processeurs et de la taille de leur chemin de données permet de limiter la surface silicium, au détriment de la capacité de calcul mais au profit de l'intégration de mémoires de travail. La Table 5.10 résumé les caractéristiques des différentes tuiles de calcul qui composent cette instance de l'architecture.

Bilan architectural Au total, l'instance proposée de l'architecture *eISP* intègre 5 tuiles de calcul, pour un total de 64 processeurs *SplitWay*, soit 32 GOPs à 250 MHz. La surface totale de cette instance est estimée à 1.07 mm^2 pour une consommation électrique de 135 mW à 250 MHz. Les capacités de mémorisation de l'ensemble des mémoires de travail de cette architecture *eISP* représentent 163 kbits au total. Avec 30 GOPs/mm^2 cette instance présente des performances inférieures à celle présentée précédemment, toutefois une telle capacité de calcul permet le portage de chaînes complexes de traitement d'image, d'autant que des mémoires LUTs peuvent être exploitées pour réaliser des traitements nécessitant un nombre important de cycles processeurs.

Notons que les résultats en consommation avec 237 MOPs/mW restent comparables à ceux de l'instance obtenue précédemment. De plus, les ressources correspondant aux éléments dédiés de l'architecture sont estimés à 70 GOPs, ce qui permet d'atteindre plus de 100 GOPs/mm^2 en capacité de calcul équivalente, soit plus de 750 MOPs/mW. Bien que les performances affichées soient élevées, elles sont nettement en deçà de celles obtenues pour la précédente instance. Ceci est principalement dû à la présence de mémoires de travail qui nécessitent de la surface silicium alors inutilisable par des ressources de calcul.

Conclusion de la création d'instance d'eISP

Après avoir appréhendé les techniques de programmation de l'architecture *eISP* par l'exemple, nous avons démontré la modularité du modèle architecturale d'*eISP*. En effet, une infinité d'instances peuvent être rapidement générées à l'aide de l'outil *geneSP* que nous avons conçu à cette fin, avant la conception du circuit en fonction des besoins des intégrateurs. Cette modularité nécessite aussi une étape de dimensionnement complexe qui sort du cadre de cette étude, ainsi que d'Adéquation-Algorithme-Architecture. En effet, l'organisation optimale des ressources de calcul, de mémorisation etc. pour des contraintes en surface et une consommation données dépendant des algorithmes destinés à être portés, donc du matériel d'acquisition, mais aussi des besoins de l'intégrateur.

La souplesse de la génération automatique des tuiles de calcul en VHDL synthétisable a été appliquée à trois cas. Nous avons d'abord proposé une solution *Vid'eISP.1* qui répond aux contraintes de surface et de consommation fixées par le cahier des charges tout en étant flexible et assurant la reconstruction de flux vidéo HD 1080p en temps réel. Celle-ci assure 40 GOPs effectif à 250 MHZ avec ses 80 Processeurs *SplitWay* sur une surface de 2,17 mm^2 pour une consommation de 295 mW. Ensuite nous avons proposé une architecture maximisant la capacité de calcul sur une surface de l'ordre du millimètre carré, celle-ci développe 55 GOPs effectifs pour 159 mW de consommation électrique à 250 MHz. Enfin, nous avons proposé une instance plus modulaire de surface identique qui développe 32 GOPs effectifs, toujours à 250 MHz.

5.4 Positionnement par rapport à l'état de l'art

Ce chapitre nous a permis d'appréhender la complexité de l'évaluation de la capacité de calcul de l'architecture *eISP*. En effet, les processeurs *SplitWay* ne représentent qu'une part des ressources de calcul réellement exploitées au sein d'une tuile de calcul. Nous avons choisi d'utiliser cette métrique comme référence (capacité de calcul effective) puisqu'il s'agit là des ressources réellement accessibles par le programmeur. Nous avons aussi proposé une mesure de la capacité de calcul globale afin de tenir compte des fonctionnalités des différents composants.

La flexibilité et l'extensibilité (*scalabilité)* est quant à elle difficilement quantifiable et seule une appréciation qualitative permet de l'estimer. Les critères de comparaison des architectures de la capacité de calcul par unité de surface en GOPs/mm^2, et par unité de consommation en mW/mm^2. La surface silicium et la consommation électrique globale sont aussi présentées.

La Table 5.11 présente les résultats obtenus pour les trois instances de l'architecture *eISP* (*Vid'eISP.1*, et *eISP* limitées à un millimètre carré de surface silicium) définies dans la section précédente. Pour chaque implémentation, et une sélection des travaux présentés dans l'état de l'art, les métriques de capacité de calcul par unité de surface, ainsi que par unité de consommation sont calculées Pour cela, les surfaces présentées sont toutes extrapolées en technologies 65 nm lorsqu'elles sont communiquées à partir d'autres technologies. L'extrapolation des consommations n'étant pas réalisable, puisque dépendant fortement des évolutions technologiques, elles sont présentées dans leur technologie d'origine, le cas échéant précisée. Cette impossibilité d'extrapoler les consommations fausse légèrement le calcul lorsque des technologies très éloignées sont utilisées. Cette table présente, à titre indicatif, le niveau de flexibilité et de *scalabilité* de chacune des architectures, elles sont évalués qualitativement entre 0 et 5. Cette table offre ainsi un aperçu du paysage des architectures embarquées potentiellement apte au traitement d'image embarqué.

A partir de la table 5.11, nous avons calculé la capacité de calcul par unité de surface et unité de consommation, ce qui permet de regrouper ces critères en une seule métrique. Les résultats présentés en Table 5.12 pour cinq des architectures les plus performantes au regard de cette métrique. Il apparaît que l'architecture *Hive'Flex* présente des résultats comparables à l'architecture *eISP* avec 173 MOPs/mW/MHz alors que nos différentes instances se placent entre 125 et 336 MOPs/mW/Mhz. Toutefois cette architecture ne respecte pas notre cahier des charges, elle dépasse d'un facteur deux le budget silicium alloué et de 100 mW la consommation.

Conclusion du positionnement par rapport à l'état de l'art

L'architecture *eISP*, avec les différentes instances que nous proposons, se situe au dessus de l'état de l'art en termes de performances de calcul par unité de surface et de consommation. L'architecture *Hive'Flex*, qui est la plus performante de l'état de l'art au regard de ces critères et dépassée par l'architecture *eISP* proposée ici.

Il est à noter qu'une étude comparative sur les temps de traitement par application peut être réalisée. Toutefois, celle-ci est inévitablement faussée par le caractère temps réel de l'architecture *eISP* dont la fréquence de fonctionnement est adaptée à la durée du programme à exécuter. Ainsi le temps d'exécution d'un algorithme de traitement d'image qui s'exécute en temps réel, à 25 images par seconde, est systématiquement proche du $\frac{1}{25}$ s. Dans un but d'évaluation des performances calculatoires, la fréquence maximale peut être utilisée, mais les résultats en consommation s'en trouveraient inévitablement dégradés.

Conclusion

Ce chapitre a permis de déterminer que l'essentiel de la surface silicium utilisé par *eISP* est alloué aux gestionnaires de voisinages et aux processeurs, dans une proportion qui dépend des choix réalisés par le concepteur. En effet, le modèle de l'architecture *eISP* permet une multitude de configurations matérielles qu'il convient de déterminer en fonction des ressources disponibles et des applications pour bénéficier d'une structure optimale. Pour explorer l'espace de conception, obtenir différentes instances de l'architecture, nous avons conçu *geneSP* qui permet de générer une architecture en VHDL synthétisable pour différentes technologies cibles ASIC, FPGA en fonction de paramètres choisis par le concepteur (nombre de processeurs etc.) Cet outil nous a permis de démontrer la flexibilité du modèle architectural au travers de multiples cas d'utilisation. L'architecture ainsi proposée est en mesure de s'adapter aux contraintes qui lui sont imposées.

La détermination de l'organisation optimale des tuiles de calcul est une problématique complexe qui dépasse le cadre de ces travaux. Définir le nombre de processeurs, d'éléments de mémorisation, la taille des chemins de données ou encore choisir les opérateurs les plus adaptés aux applications ciblées et aux contraintes fixées n'est qu'un exemple des choix qui sont permis par le modèle architectural et l'outil de génération automatique de l'architecture *geneSP*.

Deux approches d'estimation de la capacité de calcul globale de l'ensemble de l'architecture sont proposées, la première en tenant compte des fonctionnalités de tous les composants des tuiles de calcul, la seconde en tenant uniquement compte des processeurs de calcul *SplitWay*. Cette méthode sous-estime les capacités de calcul de l'architecture mais offre la mesure exacte du nombre d'opérations effectivement disponibles pour l'exécution du programme. C'est pourquoi, cette seconde méthode de mesure que nous utilisons pour quantifier la capacité de calcul de l'architecture *eISP*.

TAB. 5.11: COMPARAISON DE DIFFÉRENTES INSTANCES D'*eISP* AVEC DIFFÉRENTES ARCHITECTURES DE L'ÉTAT DE L'ART.

	Architecture	Commentaires	Capacité de calcul	Surface silicium	Conso. électrique	Perf. MOPs/mW	Perf. GOPs/mm^2	Scalabilité (0 à 5)	Flexibilité (0 à 5)
eISP	*Vid'eISP.1*	Instance générique *Vid'eISP.1*	40 GOPs	2,17 mm^2	295 mW	136	18	4	4
	eISP perf.	[5.3.3.1 page 143] Instance pour capacité de calcul	55 GOPs	1,03 mm^2	159 mW	345	53	4	4
	eISP flex.	[5.3.3.2 page 144] Instance pour la flexibilité	32 GOPs	1,07 mm^2	135 mW	237	30	3	3
Reconfigurable & Dédiées	*Zhou*	[Zhou 2003] Amélioration VGA 30 fps	NA	18 mm^2	< 300 mW$_{250\ nm}$	NA	NA	0	0
	v-MP2000	[Videantis Inc. 2007] Comp./Décomp.HD 1080p	NA	1,2 mm^2	15 mW$_{65\ nm}$	NA	NA	0	0
	v-MP4180HD	[Videantis Inc. 2008] Comp./Décomp.HD 1080p	NA	7,9 mm^2	102 mW $_{65\ nm}$	NA	NA	0	0
	OMAP	[Shi 2003] Plate-forme programmable + composants dédiés	NA	NA	4 W	NA	NA	2	3
	CRISP	[Chen 2008a] Amélioration HD 1080p HD 1080p	NA	1 mm^2	218 mW$_{180\ nm}$	NA	NA	2	2
	RICA	[Khawam 2008] Reconfigurable/programmable HD 1080p	NA	1,9 mm^2	600 mW$_{90\ nm}$	NA	NA	2	4
Programmables	*388VDO*	[Tensilica Co. 2007] Résolution D1	NA	4 mm^2	60 mW	NA	NA	1	2
	IMAP	[Kyo 2005] HD 1080p possible	100 GOPs	15 mm^2	> 1 W	<100	<7	3	3
	Imagine	[Khailany 2008] HD 1080p possible	512 GOPs	38 mm^2	> 10 W$_{130\ nm}$	< 51	< 13	5	4
	FIESTA	[Arakawa 2008] HD 1080p supporté	512 GOPs	152 mm^2	782 mW	654	3	2	4
	Tesla	[Lindholm 2008] GPU GeForce HD 1080p possible	500 GOPs	250 mm^2	150 W	3	2	2	4
	iVisual	[Cheng 2008] Reconnaissance de forme	76,9 GOPs	10 mm^2	374 mW$_{180\ nm}$	205	8	2	2
	SIMPil	[Gentile 2005] HD 1080p possible	1,45 TOPs	150 mm^2	2,8 W$_{100\ nm}$	517	10	2	3
	Xetal	[Abbo 2008] HD 1080p possible	107 GOPs	30 mm^2	600 mW$_{90\ nm}$	178	4	3	4
	Hive'flex	[Beric 2006] HD 1080p possible	135 GOPs	5 mm^2	600 mW$_{90\ nm}$	225	27	4	4

TAB. 5.12: COMPARAISON DE DIFFÉRENTES INSTANCES D'*eISP* AVEC LES CINQ ARCHITECTURES REPRÉSENTATIVES DE L'ÉTAT DE L'ART.

Architecture	Capa. calcul	Surface	Conso.	MOPs/mW	GOPs/mm^2	MOPs/mW/mm^2
Vid'eISP.1	40 GOPs	2,17 mm^2	295 mW	136	18	**63**
*eISP*perf.	55 GOPs	1,03 mm^2	159 mW	345	53	**336**
*eISP*flex.	32 GOPs	1,07 mm^2	135 mW	237	30	**222**
Xetal	107 GOPs	30 mm^2	600 mW$_{90\ nm}$	178	4	6
IMAP	100 GOPs	15 mm^2	> 1 W	<100	<7	7
Tesla	500 GOPs	250 mm^2	150 W	3	2	13
iVisual	76,9 GOPs	10 mm^2	374 mW$_{180\ nm}$	205	8	21
Hive'flex	135 GOPs	5 mm^2	600 mW$_{90\ nm}$	225	27	**45**

Dans le cadre de l'application au traitement de l'image sur téléphone portable nous proposons trois instances de l'architecture *eISP*. L'instance *Vid'eISP.1* présente une surface silicium de l'ordre de 2 mm^2 et répond aux cahier des charges établi, tant en termes de contraintes que de fonctionnalités puisqu'elle consomme 295 mW (contre notre limite de 500 mW) pour 40 GOPs à 250 MHz. Cette capacité de calcul permet l'exécution des traitements de reconstruction et d'amélioration d'image décrit dans le premier chapitre. Deux instances dont la surface est de l'ordre du millimètre carré ont été proposées, l'une privilégiant la capacité de calcul et l'autre la flexibilité. Leur consommation électrique est inférieure à 200 mW pour une capacité de calcul de 32 et 55 GOPs.

Ces trois propositions d'instances sont finalement confrontées à l'état de l'art. Nous avons choisi d'estimer les capacité de calcul par unité de surface et de consommation comme métrique pour une comparaison objective. Cette confrontation montre que l'architecture *eISP* dépasse les meilleurs composants actuels, et notamment l'architecture *Hive'Flex*.

Conclusions et perspectives

Conclusions de cette étude

LE TRAITEMENT d'amélioration de l'image en sortie de capteur bas coût est indispensable. L'étude de l'état de l'art a montré que des solutions existent mais elles sont souvent figées. L'objectif de cette étude était de proposer une architecture de calcul programmable destinée au traitement d'image en sortie de capteur sur dispositifs mobiles. De fait, cette architecture doit répondre à plusieurs contraintes, qui ont guidé toute cette étude. La première est la surface silicium limitée à deux à trois millimètres carrés avec les technologies d'intégration actuelles (65 nm), ce qui interdit l'utilisation de mémoire d'image. La seconde est la consommation électrique ne devant pas dépasser quelques centaines de milliwatts, ce qui ne permet pas l'utilisation de processeurs de calcul généralistes. La troisième est la flexibilité de l'architecture qui doit être en mesure de supporter une grande variété d'algorithmes. Enfin, sa capacité de calcul doit être suffisante pour reconstruire des flux vidéo en temps réel HD 1080p (1920×1200).

Nous avons d'abord réalisé une analyse au niveau opérateur et indépendante de l'architecture, des différents algorithmes représentatifs de la chaîne de reconstruction et d'amélioration d'image. Pour réaliser cette analyse nous avons développé la suite d'outils MAsS d'analyse dynamique de code à partir de leurs graphes d'exécution sur jeu de données réel. Cette analyse a permis de mettre en évidence que la capacité de calcul globale nécessaire à l'exécution de l'ensemble d'une chaîne de reconstruction d'image peut dépasser plusieurs dizaines de milliards d'opérations par seconde en fonction des algorithmes qui la composent. Sur ce volume important d'opérations, environ un tiers en moyenne est réellement destiné au calcul effectif des résultats des traitements, tandis que plus de la moitié des opérations est utilisée pour gérer l'accès aux données, le reste étant dédié au contrôle. Sur la base de ce constat nous avons choisi de séparer ces trois tâches afin de les réaliser en parallèle, et tout spécialement pour ce qui concerne la préparation des données à traiter. Cette étude algorithmique a mis en évidence les autres formes de parallélisme communes aux algorithmes de la chaîne de reconstruction d'image. Elle nous a aussi permis de déterminer avec précision quelles opérations sont utilisées (dynamique, proportion..), ainsi que le format des données susceptibles d'être manipulées, comme les images brutes, ou composites. Les résultats de cette étude nous conduisent à proposer une architecture de calcul parallèle extensible.

L'architecture *eISP* ainsi conçue est composée de tuiles de calcul programmables et autonomes qui sont interconnectées les unes aux autres avec un bus de type TDMA qui limite les interconnexions. Ce bus permet de transmettre les flux de données que les tuiles de calcul doivent traiter ainsi que leurs résultats. Il assure aussi la synchronisation des tuiles de calcul pouvant fonctionner à des fréquences différentes. Chacune d'entre elles est constituée d'un gestionnaire de voisinages qui a pour fonction de reconstituer les données issues du flux sous la forme de voisinages de pixels – dont la taille est paramétrable par le programmeur – directement accessibles par les processeurs de calcul *SplitWay* spécialement conçus.

Il s'agit de processeurs VLIW deux voies à jeu d'instructions orthogonal pour un compromis entre complexité architecturale et temps d'exécution des programmes. Les opérateurs et la file de registres des processeurs sont mutualisés entre les deux voies, afin de réduire la surface silicium. Les opérateurs intégrés sont des opérateurs arithmétiques et logiques entiers signés usuels, tandis qu'un plan mémoire est associé à chaque processeur de calcul. Chaque tuile de calcul intègre plusieurs processeurs *SplitWay* qui sont alimentés en données à traiter par le gestionnaire de voisinages. Ils fonctionnent en mode Multi-SIMD contrôlé par les données,

palliant ainsi aux déficiences de structures SIMD classiques. Ils sont connectés entre eux en anneaux, point à point, ce qui leur permet de s'échanger des données lorsque cela est nécessaire. De plus l'intégration d'extensions spécifiques a été prévue à différents niveaux de la tuile de calcul. Cette architecture a d'abord été validée post-synthèse en technologie TSMC 65 nm, avant de faire l'objet d'une étude en surface et en consommation.

Pour dimensionner l'architecture en fonction des contraintes et des ressources de calcul, nous avons d'abord déterminé quels sont les paramètres (type et taille des opérateurs, nombre de registres, nombre de processeurs, nombre de tuile) qui impactent sa surface et sa consommation. Pour cela, nous avons conçu un outil qui génère automatiquement le code VHDL synthétisable de l'architecture en fonction des paramètres qui sont donnés par l'utilisateur et de le synthétiser. Cette approche automatisée nous a facilité l'exploration de différentes configurations que l'architecture *eISP* peut prendre. En faisant varier le nombre de processeurs *SplitWay* par tuile de calcul, leur mémoire de travail, la taille de leur chemin de données et de leurs opérateurs (16 bits, 24 bits, 32 bits etc.), celles des données à manipuler (8 bits, 24 bits etc), la taille maximale des voisinages supportés etc., nous avons pu définir les caractéristiques en surface et en consommation pour différentes configurations de tuiles de calcul. Les résultats obtenus montrent qu'une instance 24 bits de processeur *SplitWay* consomme moins de 2 mW à 250 MHz pour une complexité de 6,2 kGates.

A partir de cette étude nous avons proposé une instance de l'architecture *eISP* capable de répondre aux contraintes de surface initialement fixées et sur laquelle différentes chaînes de reconstruction d'image ont été portées. Cette instance, composée de sept tuiles de calcul, délivre 40 GOPs à 250 MHz au niveau des 80 processeurs *SplitWay* 24 bits, sa surface totale en technologie TSMC 65 nm est alors de 2,17 mm^2 pour une consommation électrique estimée de 295 mW. La capacité de calcul totale équivalente atteint 100 GOPs. Afin de vérifier la viabilité de ces résultats, le « Layout » d'une tuile de calcul de 6 processeurs a été réalisé. Ces résultats dépassent les performances des architectures programmables introduites dans l'état de l'art. Ce modèle a été validé en technologie ASIC TSMC 65 nm au travers d'un exemple de processeur d'image destiné à la reconstruction de flux vidéo haute définition sur dispositif mobile.

La méthode développée tout au long de ce document pour obtenir une architecture de calcul programmable basse consommation s'avère viable. Une approche originale d'analyse des algorithmes est proposée et est utilisée pour proposer un modèle original et innovant d'architecture de calcul programmable parallèle. De plus, de par sa flexibilité, le modèle architectural développé ici peut être décliné pour s'adapter à des contraintes différentes ou pour d'autres compromis entre flexibilité et puissance. Afin de démontrer la modularité du modèle et des outils de génération automatique, différentes instances de l'architecture *eISP* ont été présentées. Nous sommes ainsi en mesure de proposer 3 instances répondant aux contraintes, l'une générique pour le traitement vidéo HD 1080P, et deux variantes privilégiant puissance de calcul ou flexibilité De même, les outils logiciels conçus dans le cadre de cette étude, notamment des outils d'analyse algorithmique et de génération automatique de l'architecture peuvent donner lieu à d'autres applications.

Perspectives ouvertes par cette étude

Cette section présente les différentes voies ouvertes par les travaux réalisés au cours de cette thèse. La première partie aborde la généralisation des outils logiciels développés au cours de cette étude. Ensuite, la seconde partie aborde quelques applications pouvant être adressées par l'architecture *eISP*. Enfin, la dernière partie présente certaines des modifications à apporter à l'architecture en vue d'augmenter sa capacité de calcul et de proposer d'autres approches architecturales.

Vers la synthèse automatique de haut niveau

Un outil intégré peut être proposé aux programmeurs ou aux concepteurs. Il peut notamment se baser sur les logiciels qui ont été développés afin de nous assister dans les choix architecturaux. Ces choix découlent directement ou indirectement de l'analyse algorithmique (capacité de calcul, nombre de voies du processeur *SplitWay*, nombre de processeurs, dynamique des opérateurs, nombre de registres, taille mémoire etc.) mais aussi des contraintes fixées par le cahier des charges.

L'ensemble de ces outils et de la méthodologie appliquée constitue une première ébauche d'outils de synthèse haut niveau. Pour réaliser un tel outil, il est d'abord nécessaire de systématiser l'analyse algorithmique, mais aussi de généraliser la génération automatique de l'architecture réalisée par *geneSP*. L'utilisation d'approches analytiques permettrait de préciser les résultats obtenus et ainsi offrir des instances de l'architecture plus à même de répondre aux contraintes.

Systématisation de la méthode d'analyse algorithmique

L'analyse algorithmique réalisée au cours de cette thèse a permis la mise au point d'une méthode d'analyse des applications basée sur un profil dynamique de leur exécution Cette méthode a été partiellement automatisée par sa traduction sous forme d'outils logiciels. Systématiser cette méthode et lui adjoindre des techniques analytiques, notamment pour le dimensionnement des opérateurs manipulant des nombres à virgule fixe est une piste de recherche importante pour la conception des architectures embarquées.

L'analyse du parallélisme au niveau instruction est basée sur le parcours du graphe d'exécution d'une application. Or, ce graphe présente un nœud par ressource utilisée (opérateur, registre, accès mémoire etc.), ce qui implique la présence de millions ou de milliards d'occurrences. La simplification de ce graphe d'exécution avant son analyse est stratégique, mais complexe, et a dû être réalisée manuellement durant cette thèse. Définir une méthode automatisable pourrait donc se révéler profitable pour la systématisation de l'analyse algorithmique. Enfin, l'attribution et l'identification des rôles des opérations, qui ont conduit à définir des tâches réalisées en temps masqué peuvent être automatisées. Pour cela une étude des approches existantes s'avère nécessaire.

Synthèse automatique de l'architecture

La méthode développée dans les outils de la suite MAsS destinés à déterminer le parallélisme au niveau instruction peut être employée afin de proposer un compilateur VLIW. Une première approche peut être efficacement réalisée, toutefois, l'intégration des spécificités du modèle Multi-SIMD nécessite une étude supplémentaire.

Dans le cadre de cette thèse, des outils logiciels au niveau architectural ont été conçus. L'outil *geneSP* consiste à générer automatique l'architecture *eISP* en fonction de paramètres donnés par l'utilisateur, ce qui rejoint la problématique d'adéquation-algorithme-architecture vue précédemment. Une partie des valeurs de ces paramètres dépend directement des applications et peut être obtenue par une systématisation de l'analyse algorithmique. Une seconde partie des valeurs de ces paramètres dépend des contraintes de surface, de consommation et de flexibilité fixées dans le cahier des charges.

Les contraintes de surface et de consommation peuvent être estimées à partir de modèles obtenus à l'aide d'outils de synthèse et de simulation bas niveau, mais il convient de préciser ces modèles. De plus, l'estimation de niveau de flexibilité de l'architecture est une problématique qu'il convient d'étudier spécifiquement. Il peut

par exemple être envisagé de déterminer quels algorithmes d'une base prévue à cet effet, peuvent être portés avec succès sur une instance donnée de l'architecture, le taux de réussite de leur portage correspondant à une métrique déterminant le niveau de flexibilité de l'instance étudiée.

Adéquation-algorithme-architecture

Une problématique complexe est l'adéquation-algorithme-architecture optimale permettant de répondre aux contraintes fixées. Il s'agit de déterminer une méthode, puis un outil logiciel, apte à proposer conjointement les implémentations algorithmiques optimales réparties sur les tuiles de calcul adaptées. Proposer des librairies de fonctions de traitement du signal optimisées au programmeur pourraient ainsi simplifier l'adéquation-algorithme-architecture.

Applications identifiées pour l'architecture

Cette sous-section présente quelques applications identifiées pour l'architecture *eISP* telle qu'elle est définie. La méthode de génération automatique permet de proposer rapidement de nombreuses variantes adaptées à différentes classes d'applications, et à différentes contraintes. Il s'agit d'applications flot de données qui rendent l'architecture *eISP* adaptée à ce type de traitement.

Applications de traitement d'image

Au cours de cette thèse, nous nous sommes attachés à adresser les applications de traitement d'image « proche pixel », mais les applications de traitement d'image plus haut niveau sont variées. La compression est un domaine d'application incontournable pouvant être ciblé par notre architecture, et qui de par sa programmabilité permet aux produits de supporter les standards à venir. D'autres applications consistent à la détection, et à l'appareillement de points d'intérêt, mais aussi à la reconstruction 3-D. Nous avons abordé certaines de ces applications comme la DCT, Détection de *Harris* [Harris 1988] au cours de ces travaux de thèse.

De plus, le traitement d'image est maintenant utilisé en automobile, en aérospatial, en télédétection, mais aussi pour la vidéo amateur ou professionnelle. Dans la plupart de ces cas, les contraintes en surface et en consommation sont nettement moindres que celles de notre cahier des charges, ce qui permet d'envisager de supporter des capacités de calcul plus importantes. Par ailleurs, le traitement vidéo rapide, à plusieurs centaines ou milliers d'images par seconde peut être adressé avec cette architecture, comme nous l'avons démontré dans le chapitre 5.

Application audio

Le traitement du signal audio numérique est aujourd'hui largement présent dans les dispositifs mobiles, et peut lui aussi être adressé par l'architecture *eISP*. Ces algorithmes de traitement du signal nécessitent généralement des accès à des voisinages à une dimension, et la gestion des métadonnées que nous proposons rend les tuiles de calcul en mesure de traiter plusieurs voies simultanément, ce qui tend à se généraliser (stéréo, 6 voies etc.) Les capacités de calcul nécessaires étant moindres, une instance de l'architecture *eISP* de taille et de consommation réduites pourraient être proposées pour ce type d'application.

Application aux télécommunications

La multiplication des standards de télécommunication dans le domaine de l'embarqué pose des problèmes similaires à ceux évoqués dans cette thèse, à savoir la limite en surface, consommation électrique et flexibilité.

Alors que les intégrateurs utilisent des composants et PIs distinctes pour les différents standards [Ojail 2008], une approche programmable telle que celle de l'architecture *eISP* permet de n'utiliser qu'un seul circuit. Dans ce cas, les tuiles de calcul sont générées avec un gestionnaire de voisinages 1-D, ce qui réduit considérablement la surface silicium du circuit par rapport au traitement d'image.

Les processeurs de calcul utilisés au sein des composants réseau bas niveau usuel (filaires, optiques ou radio-fréquences), intégrés dans les terminaux réseaux, les routeurs et les *switch* nécessitent d'importantes capacités de traitements, qui dépassent la dizaine de milliards d'échantillons par seconde [Bolaria 2008, Wheeler 2008]. L'architecture *eISP* proposée présente des similitudes avec certaines des approches proposées, ou avec les challenges à adresser [Lekkas 2003]. Dans ce domaine, la contrainte en surface silicium est plus faible qu'en téléphonie mobile. Une étude des modifications à apporter à l'architecture *eISP* pourrait permettre de proposer un tel circuit.

Application en cryptographie

Les applications de cryptographie peuvent être supportées par l'architecture *eISP*. Une étude des algorithmes peut conduire à adapter les ressources pour l'accélération de leur exécution. En effet, ces dispositifs doivent généralement répondre à des contraintes de surface et de consommation comparables à celles vues en téléphonie mobile. De plus, le niveau de flexibilité requis peut s'avérer important lorsqu'un dispositif de cryptographie doit être en mesure de supporter différents standards, actuels ou à venir. Pour cela, le portage d'algorithmes de cryptographie doit être étudié, et des opérateurs spécifiques ajoutés si cela s'avère nécessaire.

Autres applications de traitement du signal

L'architecture *eISP* telle qu'elle est définie dans ce document est potentiellement utilisable pour les applications en traitement du signal qui peuvent être réalisées en mode flot de données. Prenons par exemple les traitements dans l'espace de *Fourier*, ondelettes, ou autre. Le circuit global doit alors intégrer les transformées et leurs inverses, en plus des tuiles de calcul programmables de l'architecture *eISP*. Le processeur *SplitWay* VLIW 2 voies mutualisées associé à l'utilisation des métadonnées est particulièrement adapté au traitement des nombres complexes.

Par ailleurs, une étude doit être menée pour le portage de traitements récursifs sur une structure de calcul de type SIMD. En effet, les traitements récursifs peuvent être supportés par la version actuelle de l'architecture *eISP* dans le cas où la structure de calcul intègre une gestion complète des métadonnées et un faible nombre de processeurs (typiquement 1 à 4), ce qui limite la flexibilité de ce type de traitement.

Variations autour de l'architecture eISP et du processeur SplitWay

Nous avons développé un exemple d'application lié à la téléphonie mobile qui répond particulièrement à des spécifications contraintes en termes de surface et de consommation. D'autres domaines d'applications sont envisageables pourvu que les contraintes sur la surface et la consommation soient moindres.

Augmentation de la capacité de calcul

Nous avons prévu l'intégration de composants dédiés à plusieurs niveaux de l'architecture. Ceux-ci peuvent d'ailleurs être des processeurs de traitement à part entière si la surface silicium le permet. L'utilisation d'extensions dédiées, qui a fait l'objet d'une première étude avec un convoyeur et un trieur développés au laboratoire, permet de considérablement augmenter les capacités de calcul de l'architecture *eISP*. De plus les processeurs

SplitWay peuvent être remplacés par des composants dédiés avec un minimum de modification de l'architecture. Outre la possibilité d'intégrer des UAL en virgule flottante, le gestionnaire de métadonnées peut être utilisé pour émuler logiciellement les opérations sur des nombres à virgule flottante. Pour en déterminer la faisabilité, une étude algorithmique et architecturale est nécessaire pour déterminer quelles fonctionnalités doivent lui être ajoutées.

La fréquence est aujourd'hui limitée par l'unité EXECUTE du processeur *SplitWay*, ainsi que par son unité DECODE. Afin de proposer une solution apte à dépasser cette limite, nous avons proposé d'augmenter le nombre d'étages du pipeline du processeur. Les barrières de registres et *bypass* nécessaires à la réduction du chemin critique du processeur ont été identifiés. Un étage supplémentaire au niveau du décodeur peut être introduit, ainsi que deux au niveau de l'étage d'exécution, un en entrée et en sortie. L'étage d'exécution peut quant à lui faire l'objet d'opérateurs pipelinés.

Deux approches supplémentaires peuvent être mises en œuvre, la première consiste à utiliser des opérateurs exploitant le *Sub-Word Parallelism*. La seconde consiste à multiplier l'enchaînement des opérateurs de l'unité EXECUTE. Pour cela, il est nécessaire d'identifier avec précision quels sont les opérations les plus à même de se suivre comme par exemple placer un additionneur derrière le multiplieur. Nous avons ainsi pu déterminer en première approche qu'il s'agit d'additions et de soustractions ainsi que d'opérations de décalage. Le gain en performances obtenu par une telle solution reste encore à évaluer. Enfin la pertinence de l'utilisation de post-incrément et post-décrément automatique est à étudier.

Ré-utilisation du processeur SplitWay

Le processeur *SplitWay* présenté dans ce manuscrit a été conçu pour pouvoir être utilisé individuellement. Dans ce cas, sa surface augmente de entre 800 μm^2 et 2000 μm^2 en technologie TSMC 65 nm en fonction des besoins – changement de contexte automatique etc. En effet cette version intègre une unité FETCH ad-hoc au niveau de chaque processeur *SplitWay*. En mode SIMD, elle est mutualisée pour l'ensemble des processeurs au sein de l'unité de contrôle.

Le processeur *SplitWay* a d'abord été conçu comme un processeur de traitement du signal capable d'exécuter deux opérations par cycle, mais il peut aussi être utilisé comme micro-contrôleur. C'est d'ailleurs le cas pour l'initialisation, la configuration et le chargement des programmes au sein de l'architecture. Généraliser une telle utilisation nécessite le développement d'une suite logicielle adaptée et l'intégration d'extensions dédiées (par exemple différentes interfaces de communication). C'est par exemple le cas du processeur *AntX* tout spécialement dédié au contrôle et développé au laboratoire [Chevobbe 2008].

Vers l'assouplissement du SIMD et une architecture MIMD

Nous avons mis au point une architecture Multi-SIMD. Celle-ci présente déjà un niveau de flexibilité accru au regard des structures SIMD, toutefois il s'avère pertinent d'augmenter leur flexibilité en permettant l'exécution simultanées de plusieurs *threads* comme le propose *NVIDIA* avec la structure Single Instruction Multiple Threads (SIMT). Des techniques de changement de contexte dynamique peuvent aussi être mises en œuvre, comme cela a été le cas sur des précédents calculateurs du laboratoire [Letellier 1993]. Notre structure Multi-SIMD devient « comparable » à une architecture MIMD lorsque le nombre de mémoires programme devient égal au nombre de processeurs. Une étude est donc à mener sur la pertinence d'une architecture complètement MIMD de ce type dans le cadre de différentes applications.

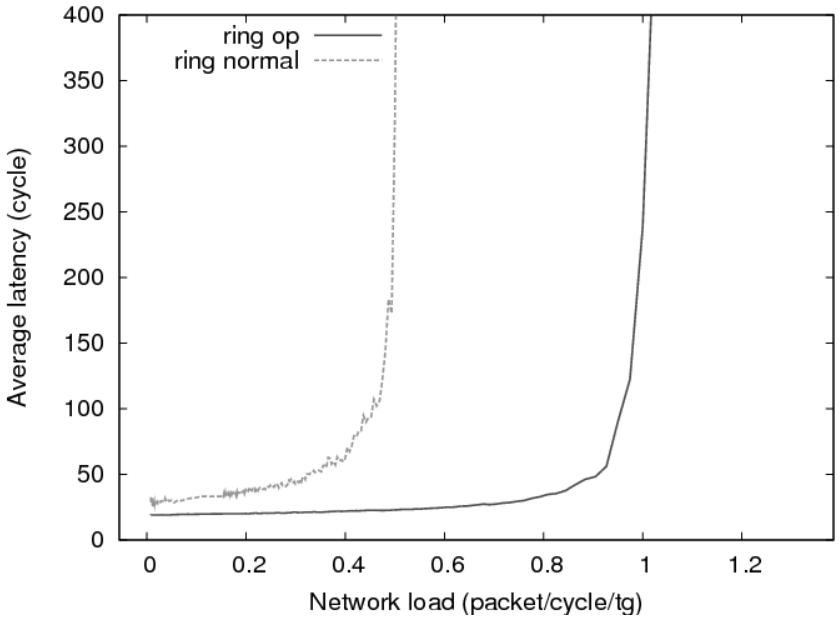


FIG. 5.16: LATENCE DU BUS TDMA IMPLÉMENTÉ EN ANNEAU, EN MODE POINT À POINT (TRAIT PLEIN) ET EN MODE PLEINEMENT CONNECTÉ (TRAIT EN POINTILLÉ).

Vers une architecture à mémoires distribuée

L'architecture proposée fonctionne actuellement en mode flot de données. Toutefois, pour une exécution efficace de certains algorithmes, comme ceux de segmentation d'image, une architecture à mémoire distribuée est à l'étude. Celle-ci ne comporte plus de gestionnaire de voisinages, mais des processeurs *SplitWay* associés à une mémoire de travail de taille plus importante. Un réseau d'interconnexion entre processeurs permet la transmission des données, des travaux réalisés au laboratoire [Collette 1992b] montre qu'il est possible d'envisager une telle structure dont l'implémentation peut faire l'objet d'une étude. Par ailleurs, la structure en tuile de calcul peut être conservée, il est alors envisageable de modifier le bus TDMA pour proposer un autre mode de communication.

Pour s'adapter à ce modèle architectural, nous avons étudié comment adapter le bus TDMA actuel pour qu'il puisse être utilisé en tant que réseau à transfert de paquets. En première approche nous proposons un réseau en anneau par transfert de paquets. Deux cas sont étudiés à l'aide d'un outil de simulation développé au laboratoire [Guerre 2009]. D'abord dans un mode de fonctionnement point à point, puis dans un mode pleinement connecté en anneau pour la transmission de messages entre différentes tuiles de calcul. La Figure 5.16 présente la latence en fonction de la charge du réseau, en trait plein le mode point à point (c'est à dire le bus TDMA segmenté entre chaque tuile). Il apparaît, sans surprise que la latence du bus réduit avec sa segmentation. Toutefois, la flexibilité réduit elle-aussi. Déterminer le compromis optimal demande donc une étude pertinente.

Conclusion sur les perspectives

De nombreuses pistes sont ouvertes par les travaux de cette thèse, nous avons d'ailleurs commencé l'exploration de certaines d'entre-elles. D'une part, les méthodes et les outils logiciels développés dans le cadre de cette thèse peuvent être complétées par ceux issus d'autres laboratoires (détermination de la dynamique des opérateurs, synthèse de haut niveau etc.). D'autre part, nous avons vu qu'une grande diversité d'instances de l'architecture *eISP* peut être générée, la dynamique des opérateurs, la taille des mots de données, la nombre de

processeurs ou des éléments de mémorisation ne sont que des exemples des paramètres qu'il est possible de faire varier en fonction du niveau de flexibilité et des contraintes en surface et en consommation recherchées. Il est ainsi possible d'utiliser cette architecture pour une grande diversité d'applications de traitement du signal en mode flux de données. Enfin, l'architecture a été conçue afin de pouvoir être adaptée pour d'autres approches que mode flux de données, comme une approche séquentielle à mémoire distribuée.

Contributions scientifiques de cette thèse

Contributions scientifiques

Cette partie du manuscrit présente les contributions scientifiques apportées dans le cadre de cette thèse.

Journaux et brevets

• [Thevenin 2009a] L'ensemble de l'architecture a fait l'objet d'un de dépôt de brevet le premier octobre 2008 intitulé « Dispositif de traitement en parallèle d'un flux de données » par *Laurent Letellier* et *Mathieu Thevenin*. Le dépôt effectif du brevet est daté du 8 juin 2009 et porte le numéro `PCT/EP2009/057033` au bénéfice du *Commissariat à l'Énergie Atomique (CEA)*.

Ce brevet couvre les différents aspects de l'architecture *eISP*développée dans le cadre de cette thèse, essentiellement :

- Le dispositif de mise à disposition d'un flux de données sous forme de matrice à des unités de calcul, il s'agit du gestionnaire de voisinages;
- Le processeur VLIW *SplitWay* dont le jeu d'opérateurs est mutualisé pour les différentes voies;
- Le support et la gestion de métadonnées et de données composites;
- Le fonctionnement en mode Multi-SIMD contrôlé par ces données composites;
- L'organisation de l'architecture en tuiles de calcul interconnectées par un bus TDMA.

Revues avec comité de lecture

• [Thevenin 2010] « Processeur vidéo programmable pour la téléphonie mobile - eISP : une architecture de calcul très basse consommation à faible empreinte silicium pour le traitement vidéo HD »
Mathieu Thevenin, Michel Paindavoine, Laurent Letellier *Techniques Des Sciences Informatiques*, numéro spécial *SympA'08*(29) éditions *Hermes Sciences*.

• Article de revue internationale Journal of Real-Time Image Processing (JRTIP) publié par *Springer* soumis, notification attendue pour Novembre 2009.

Conférences internationales avec actes

• [Thevenin 2009b] *Mathieu Thevenin, Michel Paindavoine, Laurent Letellier, Renaud Schmit et Barthélémy Heyrman*. eISP: a Programmable Processing Architecture for Smart Phone Image Enhancement.
Conference on Design and Architecture for Signal and Image Processing - DASIP 2009 du 22 au 24 Septembre 2009, Nice Sophia-Antipolis, France

• [Thevenin 2008] *Mathieu Thevenin, Michel Paindavoine, Laurent Letellier et Barthélémy Heyrman*. Embedded Processor Extensions for Image Processing.
Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series – Conference SPIE Photonics Europe 2008 du 7 au 11 Avril 2009, Strasbourg, France

Conférences nationales ou francophones avec actes

- [Mathieu Thevenin 2009b] *Mathieu Thevenin, Michel Paindavoine, Laurent Letellier et Barthélémy Heyrman*. eISP, une architecture de calcul programmable pour l'amélioration d'images sur téléphone portable. 22ème Colloque GRETSI 2009 du 8 au 11 Septembre 2009, Dijon, France

- [Mathieu Thevenin 2008] *Mathieu Thevenin, Michel Paindavoine, Laurent Letellier et Barthélémy Heyrman*. Extensions matérielles pour processeurs embarqués de traitement d'images. SYMPosium en Architectures nouvelles de machines SympA'08 du 11 au 13 Février 2008, Fribourg, Suisse

Conférences et colloques

- [Mathieu Thevenin 2009a] *Mathieu Thevenin, Michel Paindavoine, Laurent Letellier et Barthélémy Heyrman*. eISP : processeur vidéo pour la téléphonie mobile. IIIème Colloque du GDR SoCSiP du 10 au 12 Juin 2009, Paris-Orsay, France

ASIC Application Specific Integrated Circuit
ASIP Application Specific Processor
CAN Convertisseur Analogique-Numérique
CCD Charge-Coupled Device
CFA Color Filter Array
CMOS Complementary Metal Oxide Semi-Conductor
CRISP Coarse-Grained Reconfigurable Image Stream Processor
DCT Discrete Cosine Transform
DMA Direct Memory Access
DSP Digital Signal Processor
FPGA Field-Programmable Gate Array
FTM Fonction de Transfert de Modulation
GPU Graphics Processing Unit
HD Haute Définition
HDMI High Definition Multimedia Interface
IMAP Integrated Memory Array Processor
IS Instruction Set
ISP Image Signal Processor
JPEG Joint Photographic Experts Group
JRTIP Journal of Real-Time Image Processing
LCE Laboratoire de Calculs Embarqué
LUT Look-Up Table
MAC Multiplication-ACcumulation
MIMD Multiple Instruction Multiple Data
MAsS Modular Assembler Simulator
MPEG Moving Picture Experts Group
MORA Multimedia Oriented Reconfigurable Array

Multi-SIMD Multiple Single Instruction Multiple Data
NoC Network On Chip
OMAP Open Multimedia Platform
QCIF Common Intermediate Format
Perl Practical Extraction and Report Language
PI Propriété Intellectuelle
RAM Random Access Memory
RICA Reconfigurable Instruction Cell Array
RISC Reduced Instruction Set Computer
RTL Resistor Transistor Logic
SAD Sum of Absolute Differences
SIMD Single Instruction Multiple Data
SIMT Single Instruction Multiple Threads
SoC System on Chip
SPARC Scalable Processor Architecture
SXGA Super eXtended Graphics Array
TDMA Time Division Multiple Access
TI Texas Instruments
UAL Unité Arithmétique et Logique
VGA Video Graphic Array
VHDL VHSIC (Very High Speed Integrated Circuit) Hardware Description Language
VLIW Very Large Instruction Word

Bibliographie

[Aamodt 2008] Tor M. Aamodt et Paul Chow. *Compile-time and instruction-set methods for improving floating- to fixed-point conversion accuracy*. ACM Trans. Embed. Comput. Syst., vol. 7, no. 3, pages 1–27, 2008. 52

[Abbo 2008] A.A. Abbo, R.P. Kleihorst, V. Choudhary, L. Sevat, P. Wielage, S. Mouy, B. Vermeulen et M. Heijligers. *Xetal-II: A 107 GOPS, 600 mW Massively Parallel Processor for Video Scene Analysis*. Solid-State Circuits, IEEE Journal of, vol. 43, no. 1, pages 192–201, Jan. 2008. 38, 147

[Adams 1997] James E. Adams. *Design of practical color filter array interpolation algorithms for digital cameras*. In D. Sinha, editeur, Proc. SPIE Vol. 3028, p. 117-125, Real-Time Imaging II, Divyendu Sinha; Ed., volume 3028, pages 117–125, Mars 1997. 19

[Agarwal 2006] V. Agarwal, B.R. Abidi, A.F. Koschan et M.A. Abidi. *An Overview of Color Constancy Algorithms*. Journal of Pattern Recognition Research, vol. 1, no. 1, pages 42–54, 2006. 18

[Agarwal 2009] Ankur Agarwal, Cyril Iskander et Ravi Shankar. *Survey of Network on Chip (NoC) Architectures & Contributions*. Journal of Engineering, Computing and Architecture, vol. 3, 2009. 75

[Agarwala 2002] S. Agarwala, P. Koeppen, T. Anderson, A. Hill, M. Ales, R. Damodaran, L. Nardini, P. Wiley, S. Mullinnix, J. Leach, A. Lell, M. Gill, J. Golston, D. Hoyle, A. Rajagopal, A. Chachad, M. Agarwala, R. Castille, N. Common, J. Apostol, H. Mahmood, M. Krishnan, Duc Bui, Quang-Dieu An, P. Groves, L. Nguyen, N.S. Nagaraj et R. Simar. *A 600 MHz VLIW DSP*. In Solid-State Circuits Conference, 2002. Digest of Technical Papers. ISSCC. 2002 IEEE International, volume 1, pages 56–444 vol.1, 2002. 40

[Agostini 2001] Luciano Volcan Agostini, Sergio Bampi et Ivan Saraiva Silva. *Pipelined Fast 2-D DCT Architecture for JPEG Image Compression*. In SBCCI '01: Proceedings of the 14th symposium on Integrated circuits and systems design, page 226, Washington, DC, USA, 2001. IEEE Computer Society. 28

[Akuiyibo 2006] Ekine Akuiyibo. *Demosaicking using Adaptive Bilateral Filters*, winter 2006. 19

[Alleysson 2004] David Alleysson. *30 ans de démosaïckage - 30 years of demosaicing*. Traitement du signal, vol. 21 issue 6, pages 561–581, 2004. 9, 19

[Alleysson 2005] David Alleysson, Sabine Süsstrunk et Jeanny Hérault. *Linear demosaicing inspired by the human visual system*. IEEE Transactions on Image Processing, vol. 14, no. 4, pages 439–449, 2005. 19

[Alleysson 2006] David Alleysson, Laurence Meylan et Sabine Süsstrunk. *HDR CFA Image Rendering*. In EURASIP 14th European Signal Processing Conference, 2006. 20

[Angelo Bosco 2001] Massimo Mancirso Angelo Bosco. *ADAPTIVE FILTERING FOR IMAGE DENOISING*. In Consumer Electronics, 2001. ICCE. International Conference, pages 208–209, 2001. 14

[Arakawa 2008] S. Arakawa, Y. Yamaguchi, S. Akui, Y. Fukuda, H. Sumi, H. Hayashi, M. Igarashi, K. Ito, H. Nagano, M. Imai et N. Asari. *A 512GOPS Fully-Programmable Digital Image Processor with full HD 1080p Processing Capabilities*. In Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International, pages 312–615, Feb. 2008. 35, 147

[Arteris Co. 2005] Arteris Co. *A Comparison of Network On Chip and Busses – White Paper*. Rapport technique, Arteris Co., 2005. 75

[Artusi 2001] Alessandro Artusi, Christian Faisstnauer et Alexander Wilkie. *New Time-Dependent Tone mapping Algorithm*. Rapport technique TR-186-2-01-21, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria, Novembre 2001. 17

[Artusi 2002] Alessandro Artusi et Alexander Wilkie. *New Real-Time Tone mapping Algorithm*. Rapport technique TR-186-2-02-02, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria, Janvier 2002. 17

[Arvind 2008] Arvind. *RTL-to-Gates Synthesis using Synopsys Design Compiler*, 2008. 116, 123

[Atasu 2005] Kubilay Atasu, Günhan Dündar et Can Özturan. *An integer linear programming approach for identifying instruction-set extensions*. In CODES+ISSS '05: Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, pages 172–177, New York, NY, USA, 2005. ACM. 34

[Balland 2007] Bernard Balland. Optique géométrique - imagerie et instruments. PPUR presses polytechniques, 2007, 2007. 10

[Bauer 2008] L. Bauer, M. Shafique et J. Henkel. *Efficient Resource Utilization for an Extensible Processor Through Dynamic Instruction Set Adaptation*. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, vol. 16, no. 10, pages 1295–1308, Oct. 2008. 34

[Belanovic 2005] Pavle Belanovic et Markus Rupp. *Automated Floating-Point to Fixed-Point Conversion with the Fixify Environment*. In RSP '05: Proceedings of the 16th IEEE International Workshop on Rapid System Prototyping, pages 172–178, Washington, DC, USA, 2005. IEEE Computer Society. 52

[Bennett 2005] Eric P. Bennett et Leonard McMillan. *Video enhancement using per-pixel virtual exposures*. ACM Trans. Graph., vol. 24, no. 3, pages 845–852, 2005. 17

[Beric 2006] Aleksandar Beric et Carlos Alba Pinto. *HiveFlex Video VSP1 Demonstration.* In Multimedia, 2006. ISM'06. Eighth IEEE International Symposium on, pages 783–784, Dec. 2006. 39, 147

[Biancardi 2002] A. Biancardi et A. Mérigot. *Extending the data parallel paradigm with data-dependent operators*. Parallel Comput., vol. 28, no. 7-8, pages 995–1021, 2002. 33

[Bianco 2007] S. Bianco, F. Gasparini et R. Schettini. *Combining strategies for white balance*. In Digital Photography III. Edited by Martin, Russel A.; DiCarlo, Jeffrey M.; Sampat, Nitin. Proceedings of the SPIE, Volume 6502, pp. 65020D (2007)., volume 6502, Mars 2007. 19

[Biswas 2000] Prasenjit Biswas, Atsushi Hasegawa, Srinivas Mandaville, Mark Debbage, Andy Sturges, Fumio Arakawa, Yasuhiko Saito et Kunio Uchiyama. *SH-5: The 64-Bit SuperH Architecture*. IEEE Micro, vol. 20, no. 4, pages 28–39, 2000. 40

[Blinn 1998] Jim Blinn. Jim blinn's corner: dirty pixels. Morgan Kaufmann, 1998. 21

[Bolaria 2008] Jag Bolaria et Bob Wheeler. A guide to ethernet switch and phy chips. Linley Group, september 2008. 153

[Boschetti 2004] Marcos R. Boschetti, Ivan S. Silva et Sergio Bampi. *A Run-Time Reconfigurable Datapath Architecture for Image Processing Applications*. In DATE '04: Proceedings of the conference on Design, automation and test in Europe, page 30242, Washington, DC, USA, 2004. IEEE Computer Society. 33

[Bosco 2002] M. Battiato Bosco A. Mancuso et S. Spampinato. *Temporal noise reduction of Bayer matrixed video data*. In Multimedia and Expo, 2002. ICME '02. Proceedings. 2002 IEEE International Conference, volume Vol.1, pages 681–684, 2002. 14

[Bovik 2002] Alan Bovik. Handbook of image and video processing. Academic Press, 2002. 14, 17, 18, 21

[Brauer Burchardt 2004] C. Brauer Burchardt. *A Simple New Method for Precise Lens Distortion Correction of Low Cost Camera Systems*. In 26th DAGM Symposium, pages 570–577, 2004. 15

[Brost 2006] Vincent Brost, Fan Yang, Michel Paindavoine et Mathieu Thevenin. *Embedded system prototyping experience using multi-DSP VHDL model.* In Workshop Proceedings of the 2nd International Workshop on Reconfigurable Communication-centric System-on-Chips (ReCoSoC), July 2006. 40

[Brownrigg 1984] D. R. K. Brownrigg. *The weighted median filter*. Commun. ACM, vol. 27, no. 8, pages 807–818, 1984. 14

[Buchsbaum 1980] G. Buchsbaum. *A Spatial Processor Model for Object dolor Perception.* Franklin Inst., vol. 310, pages 1–26, 1980. 18

[Buyue Zhang 2008] Jan P Allebach Buyue Zhang. *Adaptive bilateral filter for sharpness enhancement and noise removal*. IEEE Transactions on Image Processing, vol. 17, 2008. 22

[Cadence DS 2006] Cadence DS. *Cadence SoC Encounter System Datasheet*, 2006. 123

[Castrorina 2006] Capra A. Castrorina, A. Corchs, S. Gasparini et F. Schettini. *Dynamic range optimization by local contrast correction and histogram image analysis*. In Consumer Electronics, 2006. ICCE '06. 2006 Digest of Technical Papers International Conference, volume 7-11, pages 309– 310, Janvier 2006. 17

[Cat 1996] Huy H. Cat, Antonio Gentile, John C. Eble, Myunghee Lee, Olivier Vendier, Young Joong Joo, D. Scott Wills, Martin Brooke, Nan Marie Jokerst et April S. Brown. *SIMPil: An OE Integrated SIMD Architecture for Focal Plane Processing Applications*. In Proc. 3rd International Conference on Massively Parallel Processing using Optical Interconnections, Maui, pages 44–52, 1996. 38

[Chatterji 2003] Sourav Chatterji, Manikandan Narayanan, Jason Duell et Leonid Oliker. *Performance Evaluation of Two Emerging Media Processors: VIRAM and Imagine*. In IPDPS '03: Proceedings of the 17th International Symposium on Parallel and Distributed Processing, page 229.1, Washington, DC, USA, 2003. IEEE Computer Society. 38

[Chattopadhyay 2006] A. Chattopadhyay, B. Geukes, D. Kammler, E. M. Witte, O. Schliebusch, H. Ishebabi, R. Leupers, G. Ascheid et H. Meyr. *Automatic ADL-based operand isolation for embedded processors*. In DATE '06: Proceedings of the conference on Design, automation and test in Europe, pages 600–605, 3001 Leuven, Belgium, Belgium, 2006. European Design and Automation Association. 34

[Chen 2000] Ting Chen, Peter Catrysse, Abbas E Gamal et Brian W. *How small should pixel size be ?* In Proceedings of SPIE, April 2000, pages 451–459, 2000. 7, 9

[Chen 2007] Tsung-Huang Chen et Shao-Yi Chien. *Cost Effective Color Filter Array Demosaicking with Chrominance Variance Weighted Interpolation*. In Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on, pages 1277–1280, May 2007. 28

[Chen 2008a] J.C. Chen et Shao-Yi Chien. *CRISP: Coarse-Grained Reconfigurable Image Stream Processor for Digital Still Cameras and Camcorders*. IEEE Trans. Circuits Syst. Video Technol., vol. 18, no. 9, pages 1223–1236, Sept 2008. 31, 147

[Chen 2008b] Pei-Yin Chen, Chih-Yuan Lien et Yi-Ming Lin. *A real-time image denoising chip*. In Circuits and Systems, 2008. ISCAS 2008. IEEE International Symposium on, pages 3390–3393, May 2008. 28

[Chen 2009] Liang-Bi Chen, Ruei-Ting Gu, Wei-Sheng Huang, Chien-Chou Wang, Wen-Chi Shiue, Tsung-Yu Ho, Yun-Nan Chang, Shen-Fu Hsiao, Chung-Nan Lee et Ing-Jer Huang. *An 8.69 Mvertices/s 278 Mpixels/s tile-based 3D graphics SoC HW/SW development for consumer electronics*. In ASP-DAC '09: Proceedings of the 2009 Asia and South Pacific Design Automation Conference, pages 131–132, Piscataway, NJ, USA, 2009. IEEE Press. 36

[Cheng 2000] HD. Cheng et H Xu. *A novel fuzzy logic approach to contrast enhancement*. Pattern Recognition, 2000. 17

[Cheng 2008] Chih-Chi Cheng, Chia-Hua Lin, Chung-Te Li, Samuel C. Chang et Liang-Gee Chen. *iVisual: an intelligent visual sensor SoC with 2790fps CMOS image sensor and 205GOPS/W vision processor*. In DAC '08: Proceedings of the 45th annual Design Automation Conference, pages 90–95, New York, NY, USA, 2008. ACM. 37, 147

[Chevobbe 2008] Stephane Chevobbe. *Spécifications du processeur AntX*. Rapport technique, CEA LIST, 2008. Rapport interne CEA LIST. 154

[Ching-Chih Weng Chen 2005] H. Chiou-Shann Fuh Ching-Chih Weng Chen. *A novel automatic white balance method for digital still cameras*. In Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium, volume 4, pages 3801– 3804, Mai23-26 2005. 19

[Chouliaras 2008] V. A. Chouliaras, V. M. Dwyer, S. Agha, J. L. Nunez-Yanez, D. Reisis, K. Nakos et K. Manolopoulos. *Customization of an embedded RISC CPU with SIMD extensions for video encoding: A case study*. Integr. VLSI J., vol. 41, no. 1, pages 135–152, 2008. 34

[Ciurea 2004] Florian Ciurea et Brian Funt. *Tuning Retinex Parameters*. Journal of the Electronic Imaging, vol. 13, pages 48–57, Janvier 2004. 17

[Clapp 2004] Matthew A. Clapp, Viktor Gruev et Ralph Etienne-Cummings. *Focal-plane analog image processing*. CMOS imagers: from phototransduction to image processing, pages 141–202, 2004. 28

[Claus 2007] Christopher Claus, Johannes Zeppenfeld, Florian Müller et Walter Stechele. *Using partial-run-time reconfigurable hardware to accelerate video processing in driver assistance system*. In DATE '07: Proceedings of the conference on Design, automation and test in Europe, pages 498–503, San Jose, CA, USA, 2007. EDA Consortium. 31

[Cok 1987] D. R. Cok. *Signal processing method and apparatus for producing interpolated chrominance values in a sampled color image signal*, 1987. 19

[Collette 1992a] T. Collette, Hassane Essafi, Didier Juvin et J. Kaiser. *SYMPATIX: a SIMD computer performing the low and intermediate levels of image processing*. In PARLE '92: Proceedings of the 4th International PARLE Conference on Parallel Architectures and Languages Europe, pages 147–161, London, UK, 1992. Springer-Verlag. 37

[Collette 1992b] Thierry Collette. *Architecture et validation comportementale en VHDL d'un calculateur parallèle dédié à la vision*. PhD thesis, Institut National Polytechnique de Grenoble - INPG, September 1992. 37, 155

[Collette 1997] T. Collette, C. Gamrat, D. Juvin, J.F. Larue, L. Letellier, R. Schmit et M. Viala. *Symphonie calculateur massivement parallèle modélisation et réalisation*. Traitement du Signal - numéro spécial, vol. 14, pages 637–644, 1997. 37

[Collette 2001] Thierry Collette. *Symphonie a pris son envol*, 2001. 37

[Compton 2002] Katherine Compton et Scott Hauck. *Reconfigurable computing: a survey of systems and software*. ACM Comput. Surv., vol. 34, no. 2, pages 171–210, 2002. 30

[Coors 2002] Martin Coors, Holger Keding, Olaf Lüthje et Heinrich Meyr. *Design and DSP implementation of fixed-point systems*. EURASIP J. Appl. Signal Process., vol. 2002, no. 1, pages 908–925, 2002. 52

[Corp. 2009] ARM Corp. *Mali: Graphics Processing Units from ARM*, 2009. 36

[Cortus 2008] Cortus. *Cortus APS3 Datasheet*, 2008. 41, 68

[Cousin 2000] J.G. Cousin, M. Denoual, D.Saillé et O. Sentieys. *Fast ASIP synthesis and power estimation for DSP applications*. In IEEE Symposium on signal processing systems SIPS'2000, pages 591–600, 2000. 3

[Dally 2001] William J. Dally, Scott Rixner et Scott Rixner. *A Bandwidth-efficient Architecture for a Streaming Media Processor*, 2001. 35

[Dasu 2002] A. Dasu et S. Panchanathan. *A survey of media processing approaches*. Circuits and Systems for Video Technology, IEEE Transactions, vol. 12, pages 633–645, 2002. 27

[David 2002] Raphaël David, Daniel Chillet, Sébastien Pillement et Olivier Sentieys. *DART: A Dynamically Reconfigurable Architecture Dealing with Future Mobile Telecommunications Constraints*. In IPDPS '02: Proceedings of the 16th International Parallel and Distributed Processing Symposium, page 108, Washington, DC, USA, 2002. IEEE Computer Society. 33

[David 2003] Raphaël David. *Architecture reconfigurable dynamiquement pour applications mobiles*. PhD thesis, Université de Rennes 1, july 2003. 33

[Devices 2007] Analog Devices. *AD6722 (Dione) GSM/GPRS – Integrated Featurephone Baseband Processor*. Product sheet, Analog Devices, Analog Devices - One Technology Way - P.O. Box 9106 Norwood, MA 02062-9106 -US, 2007. 29

[Dippel 2002] S. Dippel, M. Stahl, R. Wiemker et T. Blaffert. *Multiscale Contrast Enhancement for Radiographies: Laplacian Pyramid versus Fast Wavelet Transform*. Journal of Medical Imaging, vol. 21, no. 4, pages 343–353, April 2002. 17

[Dubois 2005] E. Dubois. *Frequency-Domain Methods for Demosaicking of Bayer-Sampled Color Images*. IEEE Signal Processing Letters, vol. 12, no. 12, pages 847–850, Décembre 2005. 20

[Dubois 2008] Jérôme Dubois. *Conception en technologie CMOS d'un Système de Vision dédié à l'Imagerie Rapide et aux Traitements d'Images*. PhD thesis, Université de Bourgogne, 2008. 28

[Ebner 2009] Marc Ebner. *Color constancy based on local space average color*. Mach. Vision Appl., vol. 20, no. 5, pages 283–301, 2009. 19

[Eijndhoven 1999] J. T. J. van Eijndhoven, F. W. Sijstermans, K. A. Vissers, E. J. D. Pol, M. J. A. Tromp, P. Struik, R. H. J. Bloks, P. Van Der Wolf, A. D. Pimentel et H. P. E. Vranken. *TriMedia CPU64 Architecture*, 1999. 40

[El Gamal 2002] A. El Gamal. *Trends in CMOS image sensor technology and design*. In Electron Devices Meeting, 2002. IEDM '02. Digest. International, pages 805–808, 2002. 9

[Elhamzi 2008] W. Elhamzi, T. Saidani, M. Atri et R. Tourki. *On hardware implementation of DCT/IDCT for image processing*. In Signals, Circuits and Systems, 2008. SCS 2008. 2nd International Conference on, pages 1–4, Nov. 2008. 28

[Eun 1998] Se Young Eun et M.H. Sunwoo. *An efficient 2-D convolver chip for real-time image processing*. In Design Automation Conference 1998. Proceedings of the ASP-DAC '98. Asia and South Pacific, pages 329–330, Feb 1998. 28

[Ezer 2000] G. Ezer. *Xtensa with user defined DSP coprocessor microarchitectures*. In Computer Design, 2000. Proceedings. 2000 International Conference on, pages 335–342, 2000. 34

[Farsiu 2006] Sina Farsiu, Michael Elad et Peyman Milanfar. *Multiframe demosaicing and super-resolution of color images*. IEEE Transactions on Image Processing, vol. 15, no. 1, pages 141–159, 2006. 20

[Félice 2009] Pierre De Félice. Histoire de l'optique. Editions de l'Armathan, 2009. 10

[Fisher 1984] J.A. Fisher. *The VLIW Machine: A Multiprocessor for Compiling Scientific Code*. Computer, vol. 17, no. 7, pages 45–53, July 1984. 57

[Fontaine 2008] S. Fontaine, S. Goyette, J.M.P. Langlois et G. Bois. *Acceleration of a 3D target tracking algorithm using an application specific instruction set processor*. In Computer Design, 2008. ICCD 2008. IEEE International Conference on, pages 255–259, Oct. 2008. 34

[Frank 2007] O. Frank, R. Katz, C.L. Tisse et H.F. Durrant Whyte. *Camera calibration for miniature, low-cost, wide-angle imaging systems*. In British Machine Vision Association Conference, 2007. 15

[Fujita 1995] Y. Fujita, N. Yamashita et S. Okazaki. *A 64 parallel integrated memory array processor and a 30 GIPS real-time vision system*. Computer Architectures for Machine Perception, 1995. Proceedings. CAMP '95, pages 242–249, Sep 1995. 35

[Gamal 2005] A. El Gamal et H. Eltoukhy. *CMOS image sensors*. IEEE Circuits and Devices Magazine, vol. 21, pages 6–20, May-June 2005. 8

[Gansner 1999] Emden R. Gansner et Stephen C. North. *An Open Graph Visualization System and Its Applications to Software Engineering*. Software - Practice and Experience, vol. 30, pages 1203–1233, 1999. 45

[Garcia-Lamont 2008] J. Garcia-Lamont, M. Aleman-Arce et J. Waissman-Vilanova. *A Digital Real Time Image Demosaicking Implementation for High Definition Video Cameras*. In Electronics, Robotics and Automotive Mechanics Conference, 2008. CERMA '08, pages 565–569, 30 2008-Oct. 3 2008. 28

[Gavrincea 2007] G.C. Gavrincea, A. Tisan, A. Buchman et S. Oniga. *Survey of wavelet based denoising filter design*. In 30th International Spring Seminar on Electronics Technology, may 2007. 15

[Gentile 2004] Antonio Gentile et D. Scott Wills. *Portable Video Supercomputing*. IEEE Transactions on Computers, vol. 53, no. 8, pages 960–973, 2004. 38

[Gentile 2005] A. Gentile, S. Vitabile, L. Verdoscia et F. Sorbello. *Image processing chain for digital still cameras based on the SIMPil architecture*. Parallel Processing, 2005. ICPP 2005 Workshops. International Conference Workshops on, pages 215–222, June 2005. 38, 147

[Gijsenij 2006] A. Gijsenij et Th. Gevers. *Color Constancy by Local Averaging and Scale Selection*. In Proceedings of the Twelfth Annual Conference of the Advanced School for Computing and Imaging, pages 141–148, Juin14-16 2006. 19

[Gil 1993] J. Gil et M. Werman. *Computing 2-D Min, Median, and Max Filters*. IEEE Trans. Pattern Anal. Mach. Intell., vol. 15, no. 5, pages 504–507, 1993. 14

[Ginhac 2008] Dominique Ginhac, Jérôme Dubois, Michel Paindavoine et Barthélémy Heyrman. *An SIMD programmable vision chip with high-speed focal plane image processing*. EURASIP J. Embedded Syst., vol. 2008, pages 1–13, 2008. 28

[Global Sources 2009] Global Sources. *Mobile phone camera modules - Mobile phones spur output growth, R&D activities in camera modules segment*. Global Sources, vol. Part 1 to 4, page NA, Mai 2009. 1, 8

[Glossner 2000] J. Glossner, J. Moreno, M. Moudgill, J. Derby, E. Hokenek, D. Meltzer, U. Shvadron et M. Ware. *Trends in compilable DSP architecture*. In Signal Processing Systems, 2000. SiPS 2000. 2000 IEEE Workshop on, pages 181–199, 2000. 41

[Gonzalez 2000] R.E. Gonzalez. *Xtensa: a configurable and extensible processor*. Micro, IEEE, vol. 20, no. 2, pages 60–70, Mar/Apr 2000. 34

[Goodacre 2005] J. Goodacre et A.N. Sloss. *Parallelism and the ARM instruction set architecture*. IEEE Computer, vol. 38, pages 42– 50, July 2005. 41

[Goodwin 2007] David Goodwin, Chris Rowen et Grant Martin. *Configurable Multi-Processor Platforms for Next Generation Embedded Systems*. Design Automation Conference, 2007. ASP-DAC '07. Asia and South Pacific, pages 744–746, Jan. 2007. 43

[Gorokhovskiy 2006] K. Gorokhovskiy, J.A. Flint et S. Datta. *Alternative color filter array layouts for digital photography*. PhD thesis, Dept. of Electron. & Electr. Eng., Loughborough Univ.;, 2006. 9

[Guerre 2009] A. Guerre, N. Ventroux ;, R. David et A. Mérigot. *Approximate-Timed Transactional Level Modeling for MPSoC Exploration: a Network-on-Chip Case Study*. In 12th Euromicro Conference on Digital System Design - Architectures, Methods and Tools (DSD - 2009), august 2009. 155

[Gunturk 2002] Y. Mersereau Gunturk B.K. Altunbasak. *Color plane interpolation using alternating projections*. In Acoustics, Speech, and Signal Processing, 2002. Proceedings. (ICASSP '02). IEEE International Conference, volume Vol.4, pages IV–3333– IV–3336, 2002. 19

[Gunturk 2005] Gunturk, B.K. Glotzbach, J. Altunbasak, Y. Schafer et R.M. R.W. Mersereau. *Demosaicking: color filter array interpolation*. Signal Processing Magazine, vol. 22 Iss. 1, pages 44–54, Janvier 2005. 19

[Gupta 2001] Maya. R. Gupta et Ting Chen. *Vector Color Filter Array Demosaicing*. In M. M. Blouke, J. Canosa et N. Sampat, editeurs, Proc. SPIE Vol. 4306, p. 374-382, Sensors and Camera Systems for Scientific, Industrial, and Digital Photography Applications II, Morley M. Blouke; John Canosa; Nitin Sampat; Eds., volume 4306, pages 374–382, Mai 2001. 19

[Hamilton 1997] J.F. Hamilton et J.E. Adams. *Adaptive color plane interpolation in single sensor color electronic camera*. US. Patent 5,629,734, 1997. 19, 24

[Hamilton 2007] John F. JR. Hamilton. *Processing color and panchromatic pixels*. US Patent Application 20070024879, 2007. 9

[Hamza 1999] Abdessamad Ben Hamza, Pedro L. Luque-Escamilla, José Martínez-Aroza et Ramón Román-Roldán. *Removing Noise and Preserving Details with Relaxed Median Filters*. J. Math. Imaging Vis., vol. 11, no. 2, pages 161–177, 1999. 14

[Hansson 2007] Andreas Hansson et Kees Goossens. *Trade-offs in the Configuration of a Network on Chip for Multiple Use-Cases*. Networks-on-Chip, International Symposium on, vol. 0, pages 233–242, 2007. 75

[Hanzo 2007] Lajos Hanzo, Peter J. Cherriman et Jürgen Streit. Video compression and communications. Wiley IEEE Press, 2007. 12

[Harris 1988] Harris et Stephans. *A Combined Corner and Edge Detector*. In Alvey Vision Conference, pages 147–152, 1988. 152

[Hartmann 2007] M. Hartmann, V. Pantazis, T. Vander Aa, M. Berekovic, C. Hochberger et B. de Sutter. *Still Image Processing on Coarse-Grained Reconfigurable Array Architectures*. In Embedded Systems for Real-Time Multimedia, 2007. ESTIMedia 2007. IEEE/ACM/IFIP Workshop on, pages 67–72, Oct. 2007. 33

[Hauser 1997] John R. Hauser et John Wawrzynek. *Garp: A MIPS Processor with a Reconfigurable Coprocessor*. In IEEE Symposium on FPGAs for Custom Computing Machines, pages 12–21, 1997. 33

[Hauser 2001] James W. Hauser et Carla N. Purdy. *Efficient Function Approximation for Embedded and ASIC Applications*. Computer Design, International Conference on, vol. 0, page 0507, 2001. 52

[Hernandez 2006] O.J. Hernandez, T. Keohane et J. Steponanko. *A Combined VLSI Architecture for Nonlinear Image Processing Filters*. In SoutheastCon, 2006. Proceedings of the IEEE, pages 261–266, 31 2005-April 2 2006. 28

[Hewlett-Packard 1998] Hewlett-Packard. *Noise Sources in CMOS Image Sensors*. Rapport technique, Hewlett-Packard Components Group, 1998. 11

[Hirakawa 2006] T.W. Hirakawa K. Parks. *Joint demosaicing and denoising*. Image Processing, 2005. ICIP 2005. IEEE International Conference, vol. 3, pages 2146– 2157 ; 309–12, Août 2006. 20

[Hoi-Jun Yoo 2008] Jun Kyong Kim Hoi-Jun Yoo et Kangmin Lee. Low-power noc for high-performance soc design. CRC Press, March 2008. 75

[Hsia 2006] Shih-Chang Hsia, Ming-Huei Chen et Po-Shien Tsai. *VLSI implementation of low-power high-quality color interpolation processor for CCD camera*. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, vol. 14, no. 4, pages 361–369, April 2006. 28

[Hung 1999] Patrick Hung, Patrick Hung, Michael J. Flynn et Michael J. Flynn. *Optimum Instruction-level Parallelism (ILP) for Superscalar and VLIW Processors*, 1999. 57

[IBM 2003] Xilinx & IBM. *PowerPC Processor Reference Guide Embedded*, 2003. 41

[Illgner 1999] K. Illgner, H.-G. Gruber, P. Gelabert, Jie Liang, Youngjun Yoo, W. Rabadi et R. Talluri. *Programmable DSP platform for digital still cameras*. In ICASSP '99: Proceedings of the Acoustics, Speech, and Signal Processing, 1999. on 1999 IEEE International Conference, pages 2235–2238, Washington, DC, USA, 1999. IEEE Computer Society. 27

[Jack 2007] Keith Jack. Video demystified : A handbook for the digital engineer. Newsnes, 2007. 21

[Jain 2001a] Manoj Kumar Jain, M. Balakrishnan et Anshul Kumar. *ASIP Design Methodologies: Survey and Issues*. In In Proceedings of the IEEE/ACM International Conference on VLSI Design. VLSI 2001, pages 76–81, 2001. 3

[Jain 2001b] M.K. Jain, M. Balakrishnan et A. Kumar. *ASIP design methodologies: survey and issues*. In VLSI Design, 2001. Fourteenth International Conference on, pages 76–81, 2001. 34

[Jin 2001] Y. Jin, L. M. Fayad et A. F. Laine. *Contrast enhancement by multiscale adaptive histogram equalization*. In A. F. Laine, M. A. Unser et A. Aldroubi, editeurs, Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, volume 4478, pages 206–213, Décembre 2001. 17

[Joshi 2006] J. Joshi, N. Nabar, R. Adyanthaya et P. Batra. *An Efficient Pipelined Architecture For Multilevel Wavelet Based Image Denoising*. In Visual Information Engineering, 2006. VIE 2006. IET International Conference on, pages 351–355, Sept. 2006. 28

[Jouppi 1989] Norman P. Jouppi. *The Distribution of Instruction-Level and Machine Parallelism and Its Effect on Performance*, 1989. 57

[Juan 2007] Esther Salamí San Juan. *Optimizing VLIW Architecture for Multimedia Application*. PhD thesis, Universitat Politècnica de Catalunya, March 2007. 57

[Kamae 2005] S. Kamae, Takahiro Irita, A. Tsukimori, S. Tarnaki, Toshihiro Hattori et Shinichi Yoshioka. *SH-mobile - low power application processor for cellular [3G cellular phones]*. In ISCAS (5), pages 5349–5352, 2005. 40

[Kao 2006] Wen-Chung Kao, Sheng-Hong Wang, ien Yang Chen et Sheng-Yuan Lin. *Design considerations of color image processing pipeline for digital cameras*. IEEE Transactions on Consumer Electronics, vol. 52, november 2006. 2

[Kapasi 2003] U.J. Kapasi, S. Rixner, W.J. Dally, B. Khailany, Jung Ho Ahn, P. Mattson et J.D. Owens. *Programmable stream processors*. Computer, vol. 36, no. 8, pages 54–62, Aug. 2003. 35

[Karungaru 2003] M. Akamatsu Karungaru S. Fukumi. *Neural networks and genetic algorithms for learning the scene illumination in color images*. In Computational Intelligence in Robotics and Automation, 2003. Proceedings. 2003 IEEE International Symposium, volume 3, pages 1085– 1089, Juillet16-20 2003. 18

[Katona 2006] Mihajlo Katona, Aleksandra Pižurica, Nikola Teslić, Vladimir Kovačević et Wilfried Philips. *A real-time wavelet-domain video denoising implementation in FPGA*. EURASIP J. Embedded Syst., vol. 2006, no. 1, pages 6–6, 2006. 28

[Kaufmann 2005] V. Kaufmann et R. Ladstädter. *Elimination of Color Fringes in Digital Photographs Caused by Lateral Chromatic Abberation*. In CIPA 2005 XX International Symposium, September 2005. 15

[Khailany 2007] B. Khailany, T. Williams, J. Lin, E. Long, M. Rygh, D. Tovey et W.J. Daly. *A Programmable 512 GOPS Stream Processor for Signal, Image, and Video Processing*. In Solid-State Circuits Conference, 2007. ISSCC 2007. Digest of Technical Papers. IEEE International, pages 272–602, Feb. 2007. 35

[Khailany 2008] B.K. Khailany, T. Williams, J. Lin, E.P. Long, M. Rygh, D.W. Tovey et W.J. Dally. *A Programmable 512 GOPS Stream Processor for Signal, Image, and Video Processing*. Solid-State Circuits, IEEE Journal of, vol. 43, no. 1, pages 202–213, Jan. 2008. 35, 147

[Khawam 2008] S. Khawam, I. Nousias, M. Milward, Ying Yi, M. Muir et T. Arslan. *The Reconfigurable Instruction Cell Array*. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, vol. 16, no. 1, pages 75–85, Jan. 2008. 33, 147

[Kittitornkun 2003] Surin Kittitornkun et Yu Hen Hu. *Programmable Digital Signal Processor (PDSP): A Survey*, august 2003. 70

[Kleihorst 2001] R.P. Kleihorst, A.A. Abbo, A. van der Avoird, M.J.R. Op de Beeck, L. Sevat, P. Wielage, R. van Veen et H. van Herten. *Xetal: a low-power high-performance smart camera processor*. In Circuits and Systems, 2001. ISCAS 2001. The 2001 IEEE International Symposium on, volume 5, pages 215–218 vol. 5, 2001. 38

[Kluge 2008] Y.-H. Kluge et H.-J. Stolberg. *V-MP2000: A flexible multi-core platform for multi-standard video applications*. In Multimedia and Expo, 2008 IEEE International Conference on, pages 1609–1610, 23 2008-April 26 2008. 28

[Kodak Eastman Co. 1995] Kodak Eastman Co. *Kodak Image Database ftp://ftp.kodak.com/www/images/pcd*, 1995. 44

[Kodak Eastman Co. 2005] Kodak Eastman Co. *CCD Image Sensor Noise Sources*. Rapport technique, Kodak Eastman Co., 2005. 11

[Kozyrakis 1999] Christoforos Kozyrakis, Christoforos Kozyrakis et Christoforos Kozyrakis. *A Media-Enhanced Vector Architecture for Embedded Memory Systems.* Rapport technique, University of California (Berkeley), 1999. 38

[Kozyrakis 2002a] Christoforos Kozyrakis et David Patterson. *Vector vs. Superscalar and VLIW Architectures for Embedded Multimedia Benchmarks.* In In Proceedings of the 35th Annual IEEE/ACM International Symposium on Microarchitecture, pages 283–293, 2002. 38

[Kozyrakis 2002b] Christoforos Kozyrakis et David Patterson. *Vector Vs. Superscalar and VLIW Architectures for Embedded Multimedia Benchmarks.* Microarchitecture, IEEE/ACM International Symposium on, vol. 0, page 283, 2002. 38

[Kyo 2001] S. Kyo, T. Koga et S. Okazaki. *IMAP-CE: a 51.2 GOPS video rate image processor with 128 VLIW processing elements*. In Image Processing, 2001. Proceedings. 2001 International Conference on, volume 3, pages 294–297 vol.3, 2001. 35

[Kyo 2005] S. Kyo, S. Okazaki et T. Arai. *An integrated memory array processor architecture for embedded image recognition systems*. Computer Architecture, 2005. ISCA '05. Proceedings. 32nd International Symposium on, pages 134–145, June 2005. 35, 147

[Lam 2005] E.Y. Lam. *Combining gray world and retinex theory for automatic white balance in digital photography*. In onsumer Electronics, 2005. (ISCE 2005). Proceedings of the Ninth International Symposium, pages 134–139, Juin14-16 2005. 19

[Land 1971] E.H. Land et J.J. McCann. *Lightness and Retinex Theory*. Journal of the Optical Society of America, vol. 61, no. 1, pages 1–11, 1971. 18

[Lane 1999] Terran Lane et Carla E. Brodley. *Temporal sequence learning and data reduction for anomaly detection*. ACM Trans. Inf. Syst. Secur., vol. 2, no. 3, pages 295–331, 1999. 14

[Lanuzza 2007] M. Lanuzza, S. Perri, P. Corsonello et M. Margala. *A New Reconfigurable Coarse-Grain Architecture for Multimedia Applications*. In Adaptive Hardware and Systems, 2007. AHS 2007. Second NASA/ESA Conference on, pages 119–126, Aug. 2007. 32

[LEE 1980] J.S. LEE. *Digital image enhancement and noise filtering by use of local statistics*. IEEE Transactions on Pattern Analysis and Machine Intelligence., vol. PAMI-2, pages 165–168, March 1980. 17

[Lee 1995] R. Lee. *Multimedia Acceleration with Subword Parallelism in Microprocessors*. Distinguished Lecture Series X, March 1995. 70

[Lee 1996] Ruby B. Lee. *Subword Parallelism with MAX-2*. IEEE Micro, vol. 16, no. 4, pages 51–59, 1996. 70

[Lee 2000] Ming-Hau Lee, Hartej Singh, Guangming Lu, Nader Bagherzadeh, Fadi J. Kurdahi, Fadi et J. Kurdahi. *Design and Implementation of the MorphoSys Reconfigurable Computing Processor*. In Journal of VLSI and Signal Processing-Systems for Signal, Image and Video Technology. Kluwer Academic Publishers, 2000. 33

[Lekkas 2003] Panos C. Lekkas. Network processors: architectures, protocols, and platforms. McGraw-Hill Professional, 2003. 153

[Leszczynski 1989] K.W. Leszczynski et S. Shalev. *A robust algorithm for contrast enhancement by local histogram modification*. In Image and Vision Computing, volume 7, pages 205–209, 1989. 17

[Letellier 1993] Laurent Letellier. *Synthèse d'images temps réel sur réseau linéaire de processeurs SIMD: algorithmes et architectures*. PhD thesis, Institut National Polytechnique de Toulouse, October 1993. 37, 154

[Li 2007] Bing Li, De Xu, Moon Ho Lee et Song-He Feng. *A Multi-Scale Adaptive Grey World Algorithm*. IEICE - Trans. Inf. Syst., vol. E90-D, no. 7, pages 1121–1124, 2007. 19

[Lin 2005] Tay-Jyi Lin, Chie-Min Chao, Chia-Hsien Liu, Pi-Chen Hsiao, Shin-Kai Chen, Li-Chun Lin, Chih-Wei Liu et Chein-Wei Jen. *A unified processor architecture for RISC & VLIW DSP*. In GLSVSLI '05: Proceedings of the 15th ACM Great Lakes symposium on VLSI, pages 50–55, New York, NY, USA, 2005. ACM. 40

[Lindholm 2008] E. Lindholm, J. Nickolls, S. Oberman et J. Montrym. *NVIDIA Tesla: A Unified Graphics and Computing Architecture*. Micro, IEEE, vol. 28, no. 2, pages 39–55, March-April 2008. 30, 36, 147

[Lopez 1998] D. Lopez, J. Llosa, M. Valero et E. Ayguade. *Widening resources: a cost-effective technique for aggressive ILP architectures*. In Microarchitecture, 1998. MICRO-31. Proceedings. 31st Annual ACM/IEEE International Symposium on, pages 237–246, Nov-2 Dec 1998. 57

[Lukac 2004] Rastislav Lukac, Karl Martin et Konstantinos N. Plataniotis. *Demosaicked image postprocessing using local color ratios*. IEEE Trans. Circuits Syst. Video Techn, vol. 14, no. 6, pages 914–920, 2004. 19

[Lukac 2005] K.N. Lukac R. Plataniotis. *Fast video demosaicking solution for mobile phone imaging applications*. Consumer Electronics, vol. 15 Iss. 2, pages 675– 681, Mai 2005. 20

[Mallon 2007] J. Mallon et P.F. Whelan. *Calibration and removal of lateral chromatic aberration in images*. Pattern Recognition Letters, vol. 28, no. 1, pages 125–135, January 2007. 15

[Malm 2007] Henrik Malm, Magnus Oskarsson, Petrik Clarberg, Jon Hasselgren, Calle Lejdfors et Eric Warrant. *Adaptive enhancement and noise reduction in very low light-level video*. In International Conference on Computer Vision, Rio de Janeiro, Brazil, 2007. 17

[Malvar 2004] R. Malvar H.S. Li-wei He; Cutler. *High-quality linear interpolation for demosaicing of Bayer-patterned color images*. In Acoustics, Speech, and Signal Processing, 2004. Proceedings. (ICASSP '04). IEEE International Conference, volume 3, pages iii– 485–8. (ICASSP '04). IEEE International Conferencce, Mai17-21 2004. 19

[Manet 2008] Philippe Manet, Daniel Maufroid, Leonardo Tosi, Gregory Gailliard, Olivier Mulertt, Marco Di Ciano, Jean-Didier Legat, Denis Aulagnier, Christian Gamrat, Raffaele Liberati, Vincenzo La Barba, Pol Cuvelier, Bertrand Rousseau et Paul Gelineau. *An evaluation of dynamic partial reconfiguration for signal and image processing in professional electronics applications*. EURASIP J. Embedded Syst., vol. 2008, pages 1–11, 2008. 31

[Mathieu Thevenin 2008] Laurent Letellier et Barthélémy Heyrman Mathieu Thevenin Michel Paindavoine. *Extensions matérielles pour processeurs embarqués de traitement d'images*. In SYMPosium en Architectures nouvelles de machines SympA'08, february 2008. 158

[Mathieu Thevenin 2009a] Laurent Letellier et Barthélémy Heyrman Mathieu Thevenin Michel Paindavoine. *eISP : processeur vidéo pour la téléphonie mobile*. In III^{ème} Colloque du GDR SoCSiP, june 2009. 158

[Mathieu Thevenin 2009b] Laurent Letellier et Barthélémy Heyrman Mathieu Thevenin Michel Paindavoine. *eISP, une architecture de calcul programmable pour l'amélioration d'images sur téléphone portable.* In 22ème Colloque GRETSI 2009, september 2009. 158

[Matkovic 1997] Kresimir Matkovic, Laszlo Neumann et Werner Purgathofer. *A Survey of Tone Mapping Techniques*. Rapport technique TR-186-2-97-12, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria, Avril 1997. 15

[Mayo 2004] Robert N. Mayo, Robert N. Mayo, Parthasarathy Ranganathan et Parthasarathy Ranganathan. Energy consumption in mobile devices: Why future systems need requirements-aware energy scale-down, volume 3164/2005. Springer, 2004. 2

[McCaffrey 2000] Nathaniel McCaffrey. *CMOS Active Pixel Sensors with Extended Dynamic Range*. In Vision Show 2000, March 2000. 16

[McKusick 1982] Susan L. Graham Peter B. Kessler Marshall K. McKusick. *gprof: a Call Graph Execution Profiler*, 1982. 45

[Menard 2008] D. Menard, R. Serizel, R. Rocher et O. Sentieys. *Accuracy constraint determination in fixed-point system design*. EURASIP J. Embedded Syst., vol. 2008, pages 1–12, 2008. 52

[Menard uary] Daniel Menard, Daniel Chillet et Olivier Sentieys. *Floating-to-fixed-point conversion for digital signal processors*. EURASIP J. Appl. Signal Process., vol. 2006, pages 77–77, uary. 52

[Menon 2007] D. Menon et G. Calvagno. *Demosaicing Based on Wavelet Analysis of the Luminance Component*. In Image Processing, 2007. ICIP 2007. IEEE International Conference on, pages II: 181–184, 2007. 20

[Mérigot 1997] Alain Mérigot. *Associative Nets: A Graph-Based Parallel Computing Model*. IEEE Trans. Comput., vol. 46, no. 5, pages 558–571, 1997. 33

[Mérigot 2008] Alain Mérigot et Alfredo Petrosino. *Parallel processing for image and video processing: Issues and challenges*. Parallel Comput., vol. 34, no. 12, pages 694–699, 2008. 41

[Meylan 2006] L. Meylan et S. Susstrunk. *High Dynamic Range Image Rendering With a Retinex-Based Adaptive Filter*. Image Processing, vol. 15, no. 9, pages 2820–2830, August 2006. 17

[Meylan 2007] Laurence Meylan, David Alleysson et Sabine Süsstrunk. *A Model of Retinal Local Adaptation for the Tone Mapping of Color Filter Array Images*. Journal of the Optical Society of America A (JOSA), vol. ?, page ?, 2007. 17

[MIPS Co. 2005] MIPS Co. *MIPS32 è Architecture For Programmers (Volumes I to III): The MIPS32 è Privileged Resource Architecture*, 2005. 41

[Montrym 2005] J. Montrym et H. Moreton. *The GeForce 6800*. Micro, IEEE, vol. 25, no. 2, pages 41–51, March-April 2005. 30, 36

[Motwani 2004] M.C. Motwani, M.C. Gadiya, R.C. Motwani et F. C. Harris. *Survey of Image Denoising Techniques*. In Proceedings of GSP 2004, pages 27–30, Santa Clara, CA, September 2004. 13

[Mucci 2005] C. Mucci, F. Campi, A. Deledda, A. Fazzi, M. Ferri et M. Bocchi. *A cycle-accurate ISS for a dynamically reconfigurable processor architecture*. Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International, vol. 4, no. 8, page 8, April 2005. 43

[Nakano 1998] N. Nakano, R. Nishimura, H. Sai, A. Nishizawa et H. Komatsu. *Digital still camera system for megapixel CCD*. Consumer Electronics, IEEE Transactions on, vol. 44, no. 3, pages 581–586, Aug 1998. 28

[Nallaperumal 2006] K. Nallaperumal, J. Varghese, S. Saudia, D. Murugan, K. Rajalakshmi et S. Alwinallwin. *An efficient approach to Salt & Pepper Impulse Noise Reduction using Iterative Impulse Detection and Correction Phases*. International Conference on Advanced Computing and Communications, 2006. 14

[Nethercote 2007] Nicholas Nethercote et Julian Seward. *How to shadow every byte of memory used by a program*. In VEE '07: Proceedings of the 3rd international conference on Virtual execution environments, pages 65–74, New York, NY, USA, 2007. ACM. 43

[Ngathe-Simo 2009] Cindy Ngathe-Simo. Adéquation-algorithme-architecture pour l'architecture de calcul eisp. Master's thesis, Université de Bourgogne, June 2009. 140

[Ojail 2008] Maroun Ojail, Stephane Chevobbe, Raphael David et Didier Demigny. *A Frequency Domain FIR Filter Implementation Method for 3G Communication Systems*. Digital Telecommunications, International Conference on, vol. 0, pages 1–5, 2008. 153

[Perri 2007] S. Perri et P. Corsonello. *VLSI implementations of efficient isotropic flexible 2D convolvers*. Circuits, Devices & Systems, IET, vol. 1, no. 4, pages 263–269, August 2007. 28

[Petschnigg 2004] Georg Petschnigg, Richard Szeliski, Maneesh Agrawala, Michael Cohen, Hugues Hoppe et Kentaro Toyama. *Digital photography with flash and no-flash image pairs*. ACM Trans. Graph., vol. 23, no. 3, pages 664–672, 2004. 17

[Piguet 2004] Christian Piguet. Low-power electronics design. CRS Press, 2004. 4, 121

[Pillement 2007] Sébastien Pillement, Olivier Sentieys et Raphaël David. *DART: A Functional-Level Reconfigurable Architecture for High Energy Efficiency (2007)*. EURASIP Journal on Embedded Systems, vol. 2008, 2007. 33

[Pinto 2006] Carlos Alba Pinto, Aleksandar Beric, Satendra Pal Singh et Sachin Farfade. *HiveFlex-Video VSP1: Video Signal Processing Architecture for Video Coding and Post-Processing*. In Multimedia, 2006. ISM'06. Eighth IEEE International Symposium on, pages 493–500, Dec. 2006. 39

[Pirsch 2003] P. Pirsch, M. Berekovic, H.-J. Stolberg et J. Jachalsky. *VLSI Architectures for MPEG-4*. In Proc. 2003 International Symposium on VLSI Technology, Systems, and Applications, pages 208–212. IEEE Press, Piscataway, NJ, 2003. 28

[Piskorski 2007] S. Piskorski, L. Lacassagne, S. Bouaziz et D. Etiemble. *Customizing CPU Instructions for Embedded Vision Systems*. In Computer Architecture for Machine Perception and Sensing, 2006. CAMP 2006. International Workshop on, pages 59–64, Aug. 2007. 34

[Polesel 2000] Andrea Polesel, Giovanni Ramponi et V. John Mathews. *Image Enhancement via Adaptive Unsharp Masking*. IEEE Trans. Image Processing, vol. 9, pages 505–510, 2000. 22

[Provenzi 2008] E. Provenzi, C. Gatta, M. Fierro et A. Rizzi. *A Spatially Variant White-Patch and Gray-World Method for Color Image Enhancement Driven by Local Contrast*. IEEE Transactions on Pattern Analysis & Machine Intelligence, vol. 30, no. 10, pages 1757–1770, October 2008. 19

[Qiang 2006] L. Qiang et N.M. Allinson. *FPGA-based Optical Distortion Correction for Imaging Systems*. In Signal Processing, 2006 8th International Conference on, volume 2, pages –, 16-20 2006. 28

[Qureshi 2005] Shehrzad Qureshi. Embedded image processing on the tms320c6000 dsp : examples in code composer studio and matlab. Springer, 2005. 15

[Raghunathan 2003] V. Raghunathan, M.B. Srivastava et R.K. Gupta. *A survey of techniques for energy efficient on-chip communication*. In Design Automation Conference, 2003. Proceedings, pages 900–905, 2003. 74

[Ramanath 2002] Rajeev Ramanath, Wesley E. Snyder, Griff L. Bilbro et William A. Sander Iii. *Demosaicking methods for bayer color arrays*. Journal of Electronic Imaging, vol. 11, pages 306–315, 2002. 2

[Ramponi 1996] G. Ramponi, N. Strobel, S. K. Mitra et T.-H. Yu. *Nonlinear unsharp masking methods for image contrast enhancement*. Journal of Electronic Imaging, vol. 5, pages 353–366, Juillet 1996. 22

[Rau 1993] B. Ramakrishna Rau et Joseph A. Fisher. *Instruction-Level Parallel Processing: History,Overview, and Perspective*. Journal of Supercomputing, vol. 7, no. 1-2, pages 9–50, Mai 1993. 57

[Ringgenberg 2006] Kyle Ringgenberg. *Power Minimization via Coding Techniques*. In 2006 Summer Research, 2006. 122

[Rixner 2000] Scott Rixner, William J. Dally, Brucek Khailany, Peter Mattson, Ujval J. Kapasi et John D. Owens. *Register Organization for Media Processing*. In Sixth International Symposium on High-Performance Computer Architecture, 2000. HPCA-6., pages 375–386, 2000. 57

[Russ 2002] John C. Russ. The image processing handbook - 4th edition. CRC Press, 2002. 18, 22

[Ryzhyk 2006] Leonid Ryzhyk. *The ARM Architecture*, 2006. 41

[Saito 2005] T. Saito et T. Komatsu. *Sharpening-demosaicking method with a total-variation-based superresolution technique*. In N. Sampat, J. M. DiCarlo et R. J. Motta, editeurs, Digital Photography. Edited by Sampat, Nitin; DiCarlo, Jeffrey M.; Motta, Ricardo J. Proceedings of the SPIE, Volume 5678, pp. 177-188 (2005)., volume 5678, pages 177–188, Février 2005. 22

[Sakaue 1995] S. Sakaue, M. Nakayama, A. Tamura et S. Maruno. *Adaptive gamma processing of the video cameras for the expansion ofthe dynamic range*. IEEE Transactions on Consumer Electronics, vol. 41, August 1995. 17, 24

[Salminen 2002] E. Salminen, V. Lahtinen, K. Kuusilinna et T. Hamalainen. *Overview of bus-based system-on-chip interconnections*. In Circuits and Systems, 2002. ISCAS 2002. IEEE International Symposium on, volume 2, pages 372–375, 2002. 75

[Salminen 2008] Erno Salminen, Ari Kulmala et Timo D. Hamalainen. *Survey of Network-on-chip Proposals*. Rapport technique, Tampere Univ., march 2008. 75

[Seo 2007] Hae Jong Seo, Priyam Chatterjee, Hiroyuki Takeda et Peyman Milanfar. *A Comparison of Some State of the Art Image Denoising Methods*. In Forty-First Asilomar Conference on Signals, Systems and Computers, 2007. 13

[Shi 2003] K. Shi, J.S.W. Song, Pallas Yang, Minh Chau, S. Aggarwal et Uming Ko. *Hierarchical timing closure methodology for OMAPè: an open multimedia application platform*. In ASIC, 2003. Proceedings. 5th International Conference on, volume 1, pages 238–241 Vol.1, Oct. 2003. 30, 147

[Silicon Hive 2008a] Silicon Hive. *HiveFlex ISP2000 Series - Image Signal Processor Databrief*. Rapport technique, Silicon Hive, 2008. 39

[Silicon Hive 2008b] Silicon Hive. *HiveFlex VSP2500 Series - Image Signal Processor Databrief*. Rapport technique, Silicon Hive, 2008. 39

[Singh 2003] Kh. Manglem Singh, Prabin K. Bora et S. Birendra Singh. *Vector median filter for removal of impulse noise from color images*. In ICECS'03: Proceedings of the 2nd WSEAS International Conference on Electronics, Control and Signal Processing, pages 1–9, Stevens Point, Wisconsin, USA, 2003. World Scientific and Engineering Academy and Society (WSEAS). 14

[Smolka 2004] B. Smolka, A. Chydzinski, 2 K. N. Plataniotis et A. N. Venetsanopoulos. *New filtering technique for the impulsive noise reduction in color images*. Mathematical Problems in Engineering, vol. 2004, 2004. 14

[Stamatis 2001] Stephan Wong Stamatis, Stephan Wong, Stamatis Vassiliadis et Sorin Cotofana. *SAD implementation in FPGA hardware*. In In Proceedings of the 12th Annual Workshop on Circuits, Systems, and Signal Processing (PRORISC2001), page http://www.isi.edu/, 2001. 28

[Stark 2000] J. Alex Stark. *Adaptive Image Contrast Enhancement using Generalizations of Histogram Equalization*. IEEE Transactions on Image Processing 9, vol. 9, pages 889–896, 2000. 17

[Steve Furber 2000] Stephen B. Furber Steve Furber. Arm system-on-chip architecture. Addison-Wesley, August 2000. 41

[Stream Processors, Inc. 2007] Stream Processors, Inc. *Storm-1 Stream Processors, SP16HP-G220 Product Brief*. Rapport technique, Stream Processors, Inc., Apr. 2007. 35

[Stretch Inc. 2007] Stretch Inc. *The S6000 Family of Processors*. Architecture white paper, Stretch Inc., 1322 Orleans Drive, Sunnyvale, CA 94089, USA, 2007. 33

[Sun 2005] Fei Sun, N.K. Jha, S. Ravi et A. Raghunathan. *Synthesis of application-specific heterogeneous multiprocessor architectures using extensible processors*. In VLSI Design, 2005. 18th International Conference on, pages 551–556, Jan. 2005. 34

[Sun 2006] F. Sun, S. Ravi, A. Raghunathan et N. K. Jha. *A Scalable Synthesis Methodology for Application-Specific Processors*. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, vol. 14, no. 11, pages 1175–1188, Nov. 2006. 34

[Synopsys 2006a] Inc. Synopsys. *Expanding the Synopsys PrimeTime Solution with Power Analysis*, 2006. 122

[Synopsys 2006b] Inc. Synopsys. *PrimePower – Full-Chip Dynamic Power Analysis for Multimillion-Gate Designs*, 2006. 122

[Synopsys 2009] Synopsys. *DesignWare Building Block IP - Documentation Overview*. Rapport technique, Synopsys, June 2009. 116

[Takahashi 2006] Hidekazu Takahashi et Kazuhiro Saito. *Sensor Device for Automatic Exposure and Automatic Focus*. US. Patent 7,221,400, vol. B1, page , Oct 2006. 16

[Talla 2004] Deepu Talla, C.-Y. Hung, Raj Talluri, F. Brill, D. Smith, D. Brier, B. Xiong et D. Huynh. *Anatomy of a portable digital mediaprocessor*. Micro, IEEE, vol. 24, no. 2, pages 32–39, Mar-Apr 2004. 27, 29, 30

[Tensilica Co. 2007] Tensilica Co. *388VDO Video DSP Product Brief*. Rapport technique, Tensilica Co., 2007. 34, 147

[Texas Instrument 1996] Texas Instrument. *CMOS Power Consumption and C_{pd} Calculation*. Rapport technique, Texas Instrument, 1996. 121

[Texas Instruments 2006] Texas Instruments. *Texas Instruments TMS320DSC21 : A High-Performance, Programmable, Single Chip Digital Signal Processing Solution to Digital Still Cameras*. Rapport technique, Texas Instruments, 2006. 29

[Thevenin 2006] Mathieu Thevenin. Adéquation algorithme architecture : étude et proposition déune méthodologie génération automatique déarchitecture multiprocesseurs vliw. Master's thesis, Université de Bourgogne, September 2006. 40

[Thevenin 2008] M. Thevenin, M. Paindavoine, L. Letellier et B. Heyrman. *Embedded processor extensions for image processing*. In Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, volume 7001, Mai 2008. 157

[Thevenin 2009a] Mathieu Thevenin et Laurent Letellier. *Dispositif de traitement en parallèle d'un flux de données*. Patent number PCT/EP2009/057033, June 2009. 157

[Thevenin 2009b] Mathieu Thevenin, Laurent Letellier, Michel Paindavoine, Renaud Schmit et Barthsélémy Heyrman. *eISP: a Programmable Processing Architecture for Smart Phone Image Enhancement*. In Proceedings of Conference on Design and Architecture for Signal and Image Processing, September 2009. 157

[Thevenin 2010] Mathieu Thevenin, Michel Paindavoine et Laurent Letellier. *Processeur vidéo programmable pour la téléphonie mobile - eISP : une architecture de calcul très basse consommation à faible empreinte silicium pour le traitement vidéo HD*. Techniques des Sciences Informatiques - numéro spécial SympA'08, 2010. 157

[Tisse 2008] C.-L. Tisse, F. Guichard et F. Cao. *Does resolution really increase image quality ?* In Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, volume 6817, Mars 2008. 9

[Tjaden 1970] G. S. Tjaden et M. J. Flynn. *Detection and parallel execution of parallel independent instructions*. IEEE Transactions on Computers, vol. C-19, no. 10, pages 889–895, 1970. 57

[Tomasi 1998] C. Tomasi et R. Manduchi. *Bilateral Filtering for gray and color images*. In Proceedings of the 1998 IEEE International Conference on Computer Vision, Bombay, India, 1998. IEEE. 14

[Tsao 2008] You-Ming Tsao, Chih-Hao Sun, Yu-Cheng Lin, Ka-Hang Lok, Chia-Jung Hsu, Shao-Yi Chien et Liang-Gee Chen. *A 26mW 6.4GFLOPS multi-core stream processor for mobile multimedia applications*. In VLSI Circuits, 2008 IEEE Symposium on, pages 24–25, June 2008. 35

[TSMC 2007] TSMC. *65/55 Nanometer Process Technology*, april 2007. 27, 68, 117

[Tufegdzic 2009] Pam Tufegdzic. *CMOS Image Sensors Continue to Gain Visible Ground*. Rapport technique, iSuppli, 2009. 1

[Turkowski 1990] Ken Turkowski. *Filters for common resampling tasks*. Graphics gems, pages 147–165, 1990. 21

[van de Weijer 2007] J. van de Weijer, A. Gijsenij et Th. Gevers. *Edge-based color constancy*. IEEE Transactions on Image Processing, vol. ?, page ?, 2007. 19

[Venkatasubramanian 2003] Suresh Venkatasubramanian. *The Graphics Card as a stream computer*. In SIGMOD-DIMACS Workshop on Management and Processing of Data Streams, 2003. 36

[Videantis Inc. 2007] Videantis Inc. *v-MP2000SD, Dual-Core Multi-Standard Video Codec IP Solution*. Rapport technique, Videantis Inc., 2007. 28, 147

[Videantis Inc. 2008] Videantis Inc. *v-MP4180HDX, Full HD 1080p Video Codec Integrated Solution*. Rapport technique, Videantis Inc., 2008. 29, 147

[Wall 1991] David W. Wall. *Limits of instruction-level parallelism*. In ASPLOS-IV: Proceedings of the fourth international conference on Architectural support for programming languages and operating systems, pages 176–188, New York, NY, USA, 1991. ACM. 57

[Wall 200] Larry Wall. Programming perl (3rd edition). O'Reilly, 200. 45

[Wang 2004] Rui Wang, Gaoming Du, Hao Li, Xiaochun Zhu et Wenfa Zhan. *VLSI design of universal color conversion circuit*. In Radio Science Conference, 2004. Proceedings. 2004 Asia-Pacific, pages 269–272, Aug. 2004. 28

[Wheeler 2008] Bob Wheeler et Jag Bolaria. A guide to 10g ethernet adapters and controller chips. Linley Group, october 2008. 153

[Woo 2007] Jeong-Ho Woo, Ju-Ho Sohn, Hyejung Kim, Jongcheol Jeong, Euljoo Jeong, Suk Joong Lee et Hoi-Jun Yoo. *A low power multimedia SoC with fully programmable 3D graphics and MPEG4/H.264/JPEG for mobile devices*. In ISLPED '07: Proceedings of the 2007 international symposium on Low power electronics and design, pages 238–243, New York, NY, USA, 2007. ACM. 36

[Woo 2008] Jeong-Ho Woo, Ju-Ho Sohn, Hyejung Kim et Hoi-Jun Yoo. *A 195 mW/152 mW Mobile Multimedia SoC With Fully Programmable 3-D Graphics and MPEG4/H.264/JPEG*. Solid-State Circuits, IEEE Journal of, vol. 43, pages 2047–2056, Sept 2008. 36

[Wu Qifa 2002] Xu Wei Wu Qifa Ting Yujing et Masayuki Tomisawa. *Achieve Low Power Design with Power Compiler*. In Synopsys Users Group (SNUG) annual conference, 2002. 123

[Wu 1991] Xiaolin Wu. *An efficient antialiasing technique*. In SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques, pages 143–152, New York, NY, USA, 1991. ACM. 21

[Wu 2006] Xiaolin Wu et Lei Zhang. *Temporal color video demosaicking via motion estimation and data fusion*. IEEE Transactions on Circuits and Systems for Video Technology, vol. 16, 2006. 20

[Yamashita 2006] T. Yamashita, H. Shimamoto, R. Funatsu, K. Mitani et Y. Nojiri. *A lateral chromatic aberration correction system for ultrahigh-definition color video camera*. In M. M. Blouke, editeur, Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, volume 6068, pages 210–217, Février 2006. 15

[yan Huo 2006] Jun yan Huo, Yi lin Chang, Jing Wang et Xiao xia Wei. *Robust automatic white balance algorithm using gray color points in images*. Consumer Electronics, IEEE Transactions, vol. 52, Iss.2, pages 541–546, Mai 2006. 18

[Ye 2000] Zhi Alex Ye, Andreas Moshovos, Scott Hauck et Prithviraj Banerjee. *CHIMAERA: A High-Performance Architecture with a Tightly-Coupled Reconfigurable Functional Unit*. In In Proceedings of the 27th Annual International Symposium on Computer Architecture, pages 225–235. ACM Press, 2000. 33

[Yoo 2007] Byeong-Gyu Nam Jeabin Lee Kwanho Kim Seung Jin Lee Hoi-Jun Yoo. *A 52.4mW 3D Graphics Processor with 141Mvertices/s Vertex Shader and 3 Power Domains of Dynamic Voltage and Frequency Scaling*. In Solid-State Circuits Conference, 2007. ISSCC 2007. Digest of Technical Papers. IEEE International, pages 278–603, 2007. 36

[Yu 2003a] W. Yu et Y. Chung. *A calibration-free lens distortion correction method for low cost digital imaging*. In International Conference on Image Processing 2003, pages I: 813–816, 2003. 15

[Yu 2003b] Zeyun Yu, , Zeyun Yu et Rajit Bajaj. *A Fast And Adaptive Method For Image Contrast Enhancement*. In in Int. Conf. on Image Processing (ICIP), Oct 2004, pages 1001–1004, 2003. 17

[Yu 2004] Wonpil Yu, Yunkoo Chung et Jung Soh. *Vignetting Distortion Correction Method for High Quality Digital Imaging*. In ICPR '04: Proceedings of the Pattern Recognition, 17th International Conference on (ICPR'04) Volume 3, pages 666–669, Washington, DC, USA, 2004. IEEE Computer Society. 14

[Zen 1998] H. Zen, T. Koizumi, H. Yamamoto et I. Kimura. *A new digital signal processor for progressive scan CCD*. Consumer Electronics, IEEE Transactions on, vol. 44, no. 2, pages 289–296, May 1998. 28

[Zhang 2007] Lei Zhang et David Zhang. *A joint demosaicking-zooming scheme for single chip digital color cameras*. Computer Vision and Image Understanding, pages 14–25, 2007. 20

[Zhou 2003] Rongzheng Zhou, Xuefeng Chen, Feng Liu, Jie He, Tiankang Liao, Yanfeng Su, Jinghua Ye, Yajie Qin, Xiaofeng Yi et Zhiliang Hong. *System-on-chip for mega-pixel digital camera processor with auto control functions*. In ASIC, 2003. Proceedings. 5th International Conference on, volume 2, pages 894–897 Vol.2, Oct. 2003. 28, 147

[Zia-ur Rahman 2004] Glenn A. Woodell Zia-ur Rahman Daniel J. Jobson. *Retinex Processing for Automatic Image Enhancement*. Journal of Electronic Imaging, vol. 13, pages 100–110, 2004. 17

Résumé : Les capteurs CMOS sont de plus en plus présents dans les produits de grande consommation comme les téléphones portables. Les images issues de ces capteurs nécessitent divers traitements numériques avant d'être affichées. Aujourd'hui, seuls des composants dédiés sont utilisés pour maintenir un niveau de consommation électrique faible. Toutefois leur flexibilité est fortement limitée et elle ne permet pas l'intégration de nouveaux algorithmes de traitement d'image. Ce manuscrit présente la conception et la validation de l'architecture de calcul eISP entièrement programmable et originale capable de supporter la vidéo HD 1080p qui sera intégrée dans les futures générations de téléphones portables.
Mots clés : Architecture; calcul; programmable; eISP; SIMD; traitement; d'image,; flux; vidéo; haute; définition; 1080p; capteur; CMOS; CCD; bayer; brutes; filtre; traitement du signal; numérique; flexible; flot; données; processeur; SplitWay ; DSP ; bus TDMA ; tuile de calcul ; 65nm

Design and Validation of a Low Power and Low Silicon Footprint Programmable Digital Image Signal Processor : Application to High Definition Video Processing on Mobile Phone.

Abstract: High definition video sensors are more and more integrated in consumer products. The images issued from these sensors need to be digitally processed before being displayed. Today, CMOS sensors are associated to dedicated components, thus maintaining the power consumption to a low level. On the opposite side, the use of dedicated components limits the flexibility of the hardware and prevents the development of platforms able to run newer image processing algorithms. This thesis describes how the design and the implementation of the eISP architecture, a programmable architecture which presents a computational efficiency, a silicon area and a power consumption suitable for the processing of 1080p HD images inside of a mobile phone.

Keywords: Programmable; computing; architecture; eISP; SIMD; Multiple; SIMD; data-flow; image; processing; video; high; definition; 1080p; bayer; sensor; CCD; CMOS; SplitWay; processor; computing; tile; meta-data; low; power; low; silicon; footprint; 65nm; low; silicon; area; high; performance; computing; digital; signal; processing; DSP ; TDMA bus