Implantation sur SoPC d'un algorithme de stabilisation d'image temps réel avec une approche multiprocesseur embarqué.

Lionel DAMEZ, Jean-Pierre DÉRUTIN, Alexis LANDRAULT, Loïc SELIER

LASMEA, UMR CNRS 6602
24 avenue des Landais, 63177 AUBIERE Cedex, France
damez@lasmea.univ-bpclermont.fr, derutin@lasmea.univ-bpclermont.fr
landrault@lasmea.univ-bpclermont.fr, selier@lasmea.univ-bpclermont.fr

Résumé – Les architectures multiprocesseurs hautement régulières sont capables de mettre en oeuvre efficacement le parallélisme présent dans la plupart des applications de traitement d'image. Les systèmes monopuces reconfigurable (SoPC) permettent au concepteur d'ajuster finement tous les éléments de l'architecture afin de la faire correspondre au mieux aux besoins de l'application visée. Ils permettent d'embarquer un nombre croissant de composants, suffisamment pour y placer maintenant une architecture multiprocesseur, avec un réseau de communication. Dans ce papier nous présentons la parallélisation et l'implantation d'un algorithme de stabilisation en temps-réel sur technologie SoPC. Notre méthode d'implémentation générale est basée sur le raffinement d'un modèle d'architecture parallèle générique pour obtenir une architecture ajustée aux besoins de l'algorithme en terme de puissance de traitement et de capacités de communication. En pratique, le concepteur choisit les composants matériels parmi un ensemble de modules paramétrables disponibles. Nous présentons des résultats de l'implantation de l'architecture matérielle et du logiciel, et les performances sur une cible SoPC Xilinx.

Abstract – Highly regular multiprocessor architectures are particularly well suited for parallel algorithms used in image processing applications. Having to Design a System on a Programmable Chip (SoPC) offers the opportunity to adjust every system component in order to meet target application needs. Integration density has increased enough to embed a whole multiprocessor architecture, along with it's communication network. In this paper we present the parallelisation and the embedding of a real-time video stabilisation algorithm on SoPC technology. Our general hardware implementation method is based upon the derivation of a generic parallel architecture in order to build an architecture meeting application processing power and communication needs. All hardware components used are readily availables parametrisable components, choosed either inside a user IP library or as commercial IPs. Hardware and Software implementation, with performance results are presented on Xilinx SoPC targets.

1 Introduction

La conception d'applications de vision temps réel embarquées impose des contraintes très sévères, qui peuvent être conflictuelles. Il faut en particulier fournir une puissance de traitement suffisante pour traiter les informations à leur rythme d'arrivée, tout en consommant le moins de place et d'énergie possible. Il est donc en général nécessaire de faire appel à des architectures spécifiquement conçues pour respecter les contraintes de l'application visée.

Les systèmes monopuces reconfigurable (SoPC) permettent au concepteur d'ajuster finement tous les éléments de l'architecture afin de la faire correspondre au mieux aux besoins de l'application visée. Ils permettent d'embarquer un nombre croissant de composants, suffisamment pour y placer maintenant une architecture multiprocesseur, avec un réseau de communication.

Les architecture multiprocesseur hautement régulières sont connues pour mettre en oeuvre efficacement le parallélisme présent dans la plupart des applications de traitement d'image. Notamment parce celles ci comportent souvent une quantité élevée de parallélisme au niveau des données.

La possibilité d'intégrer plusieurs processeurs à l'intérieur d'un même circuit reconfigurable est à l'origine d'un nombre croissant de travaux sur les architectures parallèles sur FPGA. Dans [1] est présentée une architecture SMP utilisant un crossbar partiel et une mémoire partagée avec un protocole de cohérence de cache. Dans [2] est présentée une architecture SPMD [1] interconnectée par un bus partagé utilisant une architecture mémoire hybride, mémoire interne distribuée et mémoire externe au FPGA partagée. Dans [3] est décrit le processeur soft Openfire, compatible avec le processeur Xilinx MicroBlaze[4]. Les auteurs emploient ce processeur pour former une architecture MIMD-DM [2] en topologie anneau. [5] présente une architecture de type MIMD-DM basée sur un réseau en topologie pleinement connectée.

Dans les travaux que nous présentons dans cet article, le modèle de traitement choisi (MIMD-DM) est lui aussi mis en oeuvre avec un réseau homogène et régulier de processeurs. A partir d'un modèle d'architecture générique, le choix des composants et leur paramétrage permet d'obtenir une architecture matérielle concrète, répondant aux besoins de l'algorithme en terme de puissance de traitement et de capacités de communication.

Nous traitons aussi dans cet article le problème de l'implantation d'une application de vision embarquée en choisissant un problème concret : celui de la stabilisation vidéo temps réel.

1. Simple Programme multiple Données
2. Multiple Instruction Multiple Données avec mémoire distribuée

2 Stabilisation vidéo électronique

Dans la télé-opération, ou les systèmes d'aide à la conduite, le mouvement de la structure supportant la caméra peut provoquer des mouvements parasites. Ces mouvements dus à un sol de surface chaotique, ou à des vibrations, peuvent gêner la visualisation ou l'interprétation de la séquence d'images reçues. Dans ce cas, stabiliser une séquence d'image revient à éliminer ou adoucir la composante de mouvements non intentionnels dans l'image, tout en conservant les mouvements intentionnels intacts.

Les approches de stabilisation basées sur des algorithmes de stabilisation digitale fournissent une alternative viable en terme de performances et présentent surtout l'avantage d'être mise en oeuvre avec une architecture matérielle de traitement d'image moins encombrante et souvent moins couteuse qu'une approche basée sur l'intégration de la camera dans un support de compensation électromécanique.

Parmi les travaux qui étudient à la fois les aspects architecturaux et algorithmiques du problème de stabilisation, [6] présente une rétine CMOS intelligente, l'extraction du mouvement global étant effectué à partir de mesures de déplacements locaux en périphérie des images. Nous avons développé une application de stabilisation vidéo électronique, puis nous l'avons parallélisé tout d'abord sur un cluster de station de travail [7], nous présentons ici le portage de l'algorithme sur architecture embarquée multiprocesseur.

Cette méthode se classe parmi les méthodes planaires ou encore 2D [8], il existe aussi des méthodes employant des modèles de mouvement 3D[9] ou encore 2,5D[10].

Les algorithmes de stabilisation sont constitués d'une séquence de traitements de différents niveaux, appliqués sur les images successives de la séquence vidéo. Généralement, trois modules de traitement sont présents :

- la mise en correspondance entre les images.
- l'estimation du déplacement global (au moyen du modèle de mouvement choisi).
- la correction/compensation des mouvements afin d'obtenir une séquence stable.

L'objectif de la mise en correspondance est de calculer le déplacement d'un point ou région de la scène réelle dans le plan image 2D. Ce déplacement dans le plan image est la projection du déplacement 3D de l'objet dans la scène observée. La technique employée est celle de la détection et du suivi de primitives visuelles [8], on peut aussi procéder en effectuant le calcul du flot optique [9].

La technique de détection et suivi de primitives visuelles consiste, dans un premier temps, à localiser dans l'image i des régions riches en information visuelle (fort contraste de luminance, coins, bords, etc.), et dans un deuxième temps, à retrouver ces mêmes régions à l'image $i + 1$. Pour détecter ces primitives nous employons un détecteur basé sur les ondelettes de Harr[11].

La recherche des points correspondant aux points détectés dans l'image suivante est effectuée en appliquant une fonction de corrélation (SAD - *Sum of Absolute Differences* ou SSD - *Sum of Square Differences*) entre le motif recherché (primitive), et ses potentiels correspondants dans l'image suivante.

L'estimation du mouvement global se fait en déterminant les paramètres décrivant le mouvement entre les images. Les paramètres en question dépendent du modèle de mouvement choisi. Le modèle 2D employé ici suppose une scène planaire ou presque, c'est-à-dire que tous les points mis en correspondance et utilisés dans l'estimation se trouvent à plus ou moins la même distance de la caméra. Dans ce cas, il y a trois paramètres à estimer : deux translations horizontale et verticale (Δx, Δy) et une rotation ($\Delta\theta$) autour de l'axe optique de la caméra. L'estimation du mouvement 2D est calculée par la méthode des Moindres Carrés Médians. Cette méthode est robuste aux faux appariements qui peuvent avoir lieu lors de l'étape précédente.

3 Description de l'architecture générique

Le modèle d'architecture générique utilisé est de type MIMD-DM avec des noeuds de traitement communiquant par passage de message. Ce modèle a été choisi parce qu'il est capable de mettre en oeuvre tout type de parallélisme. Le réseau d'interconnection est un réseau statique de topologie hypercube. Le choix s'est portée sur cette topologie pour ses propriétés : à la dimension D il comporte un nombre de noeuds $n = 2^D$, son diamètre est D et le nombre de liens par noeud est $= \log(n)$ avec un nombre total de liens $\frac{n}{2} \times \log(n)$. Cette topologie facilite le routage, qui peut être calculé par une simple fonction combinatoire, et permet un routage adaptatif grâce à la présence de plusieurs chemins potentiels entre deux noeuds donnés. Cette topologie est aussi intéressante en terme d'extensibilité puisque lorsque l'on double le nombre de processeurs, le nombre de liens par noeud n'augmente que d'un.

L'architecture est régulière et homogène, chaque noeud est composé des mêmes composants : un processeur par noeud, avec de la mémoire locale pour contenir l'application et les données traitées et les dispositifs de communication. Tous les composants sont choisis parmi une librairie de composants disponibles. En fonction de l'application une architecture dérivée de ce modèle se différenciera d'une autre à travers les différentes options et paramètres relatifs à la micro architecture du processeurs (unités de traitement, options d'implémentation), à la mémoire (quantité de mémoire locale, taille des tampons potentiels), et au réseau d'interconnection (type de lien entre processeur, nombre de noeuds).

En fonction des divers composants choisis et de leur paramétrage, l'architecture finale retenue pourra être une solution assez sommaire, mais aussi être une configuration beaucoup plus complexe. Dans cette étude, les choix architecturaux ont été limités à la configuration du processeur Microblaze et à un unique type de communication. Seule le type de liaison le plus simple (connexion point à point FIFO) a été retenu car il satisfait les besoins en communication de l'application pour un coût matériel relativement faible. Nous utilisions les ports d'entrées/sorties FSL (Fast Simplex Link) du Microblaze qui permettent de relier le processeur à de la logique ou à un autre processeur via une connexion point à point FIFO. Le Microblaze possède 8 entrées et 8 sorties de ce type directement connectées à ses registres, ce qui permet de mettre en oeuvre des canaux de communication full-duplex avec 8 autres processeurs.

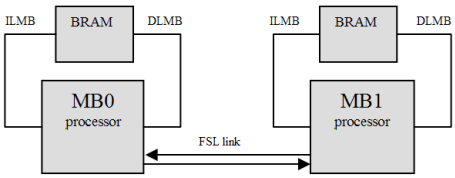

FIGURE 1 – Lien de communication point à point FSL entre 2 processeurs.

4 Implantation parallèle de l'algorithme de stabilisation

Dans [7] les caractéristiques des différents traitements de l'application on été analysés afin de concentrer l'effort de parallélisation sur les étapes amenant les gains de performances potentiellement les plus élevés. Ainsi les quatre premiers étages de traitement de la version séquentielle furent implantés en parallèle et les deux dernières étapes, ne présentant pas une complexité importante (<2%), sont implantée en mode sequentiel. Les processus parallèlisés (detection de primitives et mise en correspondance) furent ensuite portés pour notre architecture.

On peut voir sur la figure 2 le schéma d'implantation parallèle basé sur un parallélisme de donnée (images et ensuite listes de primitives) entre les N_P processeurs disponibles. Les processus (détection et mis en correspondance) ont été placés de manière statique sur les processeurs, et ordonnancés de manière à impliquer le minimum de communication possible entre les noeuds de traitement. Les différents processus sont chainés sur un même processeur de manière à ce que toutes les données intermédiaires consommées et produites restent directement disponibles au niveau local. Le principal avantage d'un tel placement/ordonnancement est d'éviter des coûts de communications supplémentaires du au transfert de données entre les processeurs. Chaque processeur traite ainsi directement les données images initiales placées dans sa mémoire locale à chaque itération, et ne communique que les résultats finaux (liste de points appareillés).

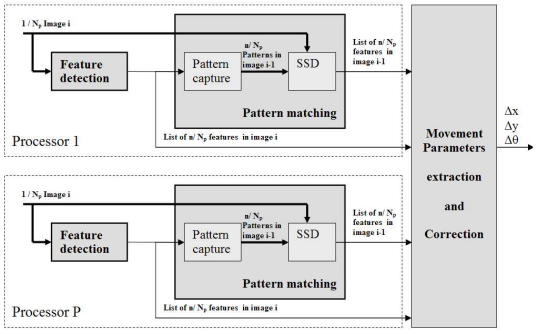

FIGURE 2 – Schéma d'implantation parallèle.

La détection est exécutée par chaque processeur p sur une bande verticale de la zone de détection globale de l'image numéro i. Cette bande est de taille $(q/N_P, r)$ et chaque processeur produit une liste de n/N_P primitives qui va être mise en correspondance dans l'étape suivante.

La mise en correspondance de primitives est l'étape la plus coûteuse en temps de traitement dans la version séquentielle. Dans la version parallèle, chaque processeur recherche dans l'image numéro $i+1$ les primitives qu'il a lui même détecté. Elles sont au nombre de (n/N_P).

5 Résultats d'implantation

Nous avons généré des architectures basées sur un type de communication FSL (FIFO 64×32 bits), avec la configuration de processeur la plus simple (aucune unité arithmétique optionnelle) et 16ko de mémoire local, en faisant varier le nombre de processeurs. Les résultats donnés en Tab.1. montrent qu'une solution à 32 processeurs peut se placer facilement dans un Virtex-4 LX200. On peut remarquer aussi que ce type de lien profite nettement des optimisations de l'architecture Virtex 5 pour implémenter des FIFOs, le cout total du système étant pratiquement réduit à celui des processeurs et de la mémoire.

TABLE 1 – ressources SoPC consommées pour 4, 8, 16, et 32 processeurs avec des connections point à point FSL sur Virtex4.

Cible	Nb μP	SLICE	FF	LUTs	BRAM
XC4VLX200	4	5%	1%	4%	9%
XC4VLX200	8	11%	2%	10%	19%
XC4VLX200	16	26%	4%	24%	38%
XC4VLX200	32	60%	7%	55%	76%

TABLE 2 – ressources SoPC consommées pour 4, 8, 16 et 32 processeurs avec des connections point à point FSL sur Virtex5.

Cible	Nb μP	SLICE	FF	LUTs	BRAM
XC5VLX330	4	3%	2%	3%	6%
XC5VLX330	8	6%	5%	6%	11%
XC5VLX330	16	12%	10%	14%	22%
XC5VLX330	32	23%	17%	28%	44%

Pour valider le flot complet de notre approche, nous avons utilisé une plateforme de test intégrant un SoPC Virtex 4 LX60. Un mécanisme matériel (actuellement en cours de développement et non décrit dans ce papier) permet d'acquérir et de découper les bandes d'images locale à chaque processeur. Les résultats pour 1 à 16 processeurs proviennent de la mesure directe sur carte. Les systèmes 32 et 64 processeurs ne pouvant pas être contenus dans notre plateforme de test, les temps d'exécution au delà de 16 processeurs on été obtenus à partir de données issues de simulation. Les temps de traitements donnés dans le tableau 3 permettent ainsi d'évaluer les accélérations obtenues entre une solution matérielle et une autre en fonction du nombre de noeuds choisi et le temps d'exécution de l'application.

TABLE 3 – Variation des paramètres d'application parallèle en fonction du nombre de noeuds.

Nombre de noeuds	1	4	8	16	32	64
H_{PE} × W_{PE}	528 × 192	144 × 192	80 × 192	48 × 192	32 × 192	24 × 192
Facteur division Image	1	3,66	6,6	11	16,5	22
Nb de motifs par noeud	64	16	8	4	2	1

TABLE 4 – Comparaison de la consommation mémoire de l'application et de celle disponible par noeud (Ko) en fonction du nombre de processeur (1 à 64).

Nombre de noeuds	1	4	8	16	32	64
taille de code	3.76	4.54	4.71	5.30	5.64	5.74
taille données image	99	27	15	9	6	4.5
Total taille Application par noeud	**102.76**	**31.54**	**19.71**	**14.30**	**11.64**	**10.24**
mémoire disponible XV4LX60	256	64	32	16	8	
mémoire disponible xv4fx140 & xv5lx330	1024	256	128	64	32	16

On peut voir dans la table 3 la taille des bandes images traitées localement par chaque noeud ainsi que le nombre de primitives détectées et ensuite mises en correspondance. La table 4 montre la taille mémoire (par noeud) nécessaire à l'exécution du programme en fonction du nombre de processeurs utilisés. La quantité de mémoire interne au circuit étant répartie entre les noeuds, la quantité de mémoire disponible par noeud diminue avec le nombre de processeurs. Ceci peut mener à une taille mémoire d'application (instruction et données) trop importante pour une configuration matérielle donnée. La taille mémoire requise pour stocker les instructions, au lieu de réduire avec le nombre de processeurs, augmente, car la complexité des communications augmente avec le nombre de noeuds. Avec le découpage en bande des données, la quantité de données image stockée en mémoire locale diminue avec le nombre de noeuds. On peut voir dans la table 4 que des système allant jusqu'à 64 processeurs peuvent être embarqués dans les plus gros SoPC Xilinx comme le virtex4FX140 ou le virtex5LX330 qui contiennent suffisamment de mémoire interne. Sur notre plateforme de test la taille mémoire disponible est suffisante pour la repartir entre 16 noeuds.

TABLE 5 – Temps d'exécution (ms) de l'application pour 1 à 64 processeurs.

Nombre de processeurs	1	4	8	16	32	64
temps de Détection	217.152	55.919	27.965	13.968	6.928	3.468
temps de mise en Correspondance	31.787	7.948	3.975	1.988	0.992	0.497
temps de Communication		0.016	0.016	0.016	0.016	0.016
temps total	**248.939**	**63.883**	**31.956**	**15.972**	**7.936**	**3.981**

TABLE 6 – Accélération de l'application pour 1 à 64 processeurs.

Nombre ode Processeur	1	4	8	16	32	64
Détection Accélération	1.00	3.88	7.78	15.55	31.34	62.72
Mise en Corr. Accélération	1.00	4.00	8.00	15.99	32.04	63.00
Accélération globale	**1.00**	**3.90**	**7.79**	**15.60**	**31.36**	**62.53**

Les temps de traitement sont donnés à partir d'un format vidéo $H_{Tot} \times W_{Tot} = 528 \times 384$, en détectant 64 primitives visuelles et en faisant une recherche dans les 6 pixels alentours pour la mise en correspondance.

On peut voir que dans la table 5 qu'avec 16 processeurs et plus le temps de traitement est bien inférieur a 40ms, ce qui laisse une marge de temps satisfaisante à l'algorithme complet (incluant les parties séquentielles) de s'exécuter à 25 images/s. A 32 processeurs, les temps de traitements sont suffisamment bas pour envisager d'employer cet algorithme comme un prétraitement d'une processus de vision de plus haut niveau. On peut observer sur la table 6 que l'accélération évolue de manière quasi linéaire avec l'accroissement du nombre de processeurs. Ces résultats s'expliquent par des temps de traitements de détection et de mise en correspondance proportionnels au nombre de primitives traitées par noeud (cf tableau 3) et par la très faible quantité de communication entre noeuds de traitement due au choix de placement/ordonnancement des processus. Notons enfin que les résultats prennent en compte uniquement les traitements implantés dans le FPGA.

6 Conclusion

Cet article présente l'implémentation d'une application réaliste de traitement d'image sur cible SoPC. Un réseau homogène comprenant un nombre élevé de processeurs sur SoPC a été réalisé et se montre efficace pour ce type d'application. Les résultats obtenus sont très intéressants, une accélération quasi linéaire et une architecture extensible permettant de faire correspondre la puissance de traitement avec la quantité de données à traiter en ajustant le nombre de processeurs.

Les travaux futurs incluent l'implantation d'algorithmes plus complexes. Ceux ci auront besoins de plus de capacités de communication. Cela nous permettra de montrer un autre type de communication (un routeur de paquets) qui n'est pas présenté dans ce papier. Notre but étant de mettre au premier plan l'architecture des processeurs et du réseau de communication pendant le raffinement de l'architecture matérielle en fonction des besoins de l'application.

Références

[1] A. Hung, W. Bishop et A. Kennings. *Symmetric multiprocessing on programmable chips made easy.* DATE '05. Proceedings of the conference on Design, Automation and Test in Europe, IEEE Computer Society, 2005.

[2] P. James-Roxby, P. Schumacher, et C. Ross. *A single program multiple data parallel processing platform for fpgas.* FCCM 2004. IEEE Symposium on Field-Programmable Custom Computing Machines, IEEE Computer Society, 2004.

[3] S. Craven et al. *Configurable soft processor arrays using the openfire processor.* MAPLD '05. Proceedings of the 8th Annual Conference on Military and Aerospace Programmable Logic Devices. Washington, DC, 2005

[4] Xilinx inc. *microblaze soft processor core* http ://www.xilinx.com.

[5] A.Patel, C. A. Madill, M. Saldana, C. Comis, R. Pomes et P. Chow. *A Scalable FPGA-based Multiprocessor.* FCCM '06. 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines. IEEE Computer Society, 2006.

[6] F. Gensolen, G. Cathebras, L. Martin et M. Robert *An Image Sensor with Global Motion Estimation for Micro Camera Module.* ACIVS '05. Conf. on Advanced Concepts for Inteligent Vision Systems. LNCS n°3708. Springer, 2005

[7] J.P. Dérutin, F.Dias, L. Damez, et N. Allezard. *Simd, smp and mimd-dm parallel approaches for real-time 2D image stabilization.* CAMP '05. Computer Architecture for Machine Perception, IEEE Computer Society, 2005.

[8] C. Morimoto. *Electronic Digital Stabilization : Design and Evaluation, with Applications.* Thèse de doctorat, University of Maryland, 1997.

[9] Z. Duric et A. Rosenfeld. *Shooting a smooth video with a shaky camera.* MVA 2003. Machine Vision and Applications n°13, 2003.

[10] Z. Zhu, G. Xu, Y. Yang et J. Jin. *Camera stabilisation based on 2.5d motion estimation and inertial motion filtering.* International Conference on Intelligent Vehicles, 1998.

[11] P. Viola et M. Jones. *Rapid object detection using a boosted cascade of simple features.* Proceedings IEEE Conf. on Computer Vision and Pattern Recognition, 2001.