Méthode d'implantation parallèle d'applications de TSI sur SoPC

Lionel DAMEZ*, Loïc SIÉLER*, Joel FALCOU**, Alexis LANDRAULT*, Jean-Pierre DÉRUTIN*

*LASMEA, UMR CNRS 6602
24 avenue des Landais, 63177 AUBIERE Cedex, France

**LRI, UMR CNRS 8623
Bât 490 Université Paris-Sud 11, 91405 Orsay Cedex France

nom@lasmea.univ-bpclermont.fr, joel.falcou@lri.fr

Résumé – Dans cet article, nous nous intéressons à une nouvelle méthodologie de conception d'architecture matérielle et logicielle de TSI sur SoPC. Le principe de notre approche est la parallélisation de l'application sur un réseau homogène de processeurs communicants. La méthodologie prévoit de guider l'utilisateur dans son choix de paramétrisation d'un réseau, constitué à partir d'une bibliothèque de composants paramétrables. Des outils d'aide à la parallélisation permettent de faciliter les choix de parallélisation de l'algorithme et guident l'utilisateur vers la définition de l'architecture adéquate. L'étape de conception materielle est fortement automatisée et permet facilement de raffiner l'architecture matérielle. Nous présentons avec la méthode, cet ensemble d'outils de conception logicielle et matérielle.

Abstract – In this paper, we introduce a new method for hardware and software architecture design dedicated to processing signal application on SoPC. This new approach is based on the concept of parallelization of the application on a homogeneous network of communicating core-processors. This approach tends to drive the designer choices for configuring the network by using a library of parameterizable IP. Specifics tools ease the parallelization of the algorithm and help the designer to converge to the suitable architecture definition. The hardware design step is strongly automatized and easily enables to refine the hardware architecture. We present here the whole method and the associated hardware and software design tools.

1 Introduction

La conception d'architectures matérielle et logicielle de caméra intelligente nécessite l'intégration de traitements complexes à proximité du capteur. Le dispositif de traitements doit d'une part fournir une puissance de calcul suffisante fonction de la fréquence du flot de pixels (contraintes temporelles), et d'autre part satisfaire des contraintes surfaciques et énérgétiques. Ces contraintes d'embarquabilité imposent en général la mise en oeuvre d'une architecture dédiée. Un autre problème est ensuite la difficulté d'étendre une telle architecture ou de pouvoir la faire évoluer facilement. Classiquement ce problème est traité par une approche co-design (partionnement hardware/software). Afin de pouvoir proposer une architecture adéquate à chaque nouvelle évolution des algorithmes, un cycle rapide de conception/implantation/validation est nécessaire. Dans cet article, nous proposons une approche originale basée sur la parallélisation de l'application et son placement sur un réseau homogène à topologie fixe de processeurs communicants embarqué sur SoPc. L'approche est basée sur l'utilisation d'outils d'aide à la parallélisation et d'outils (standards ou dédiés) de génération d'architecture matérielle à partir de bibliothèque d'IP (standards ou dédiés). Le modèle général de traitement parallèle choisi est de type Multiple Instruction Multiple Data avec mémoire distribuée (MIMD-DM) et des communications par passage de messages.

Nous présentons successivement dans la section 2 la problématique que nous adressons ; dans la section 3, les solutions répondant à notre approche ; dans la section 4, l'état d'avancement du projet pour enfin conclure et donner les perspectives.

2 Problématique

2.1 Qu'est ce que la parallélisation d'un algorithme ?

Classiquement, le passage d'une version séquentielle d'un algorithme à une version parallèle s'appuie sur l'exploitation des sources de parallélisme de la version séquentielle. Cette mise en oeuvre utilise les formes récurrentes :

- de parallélisme de données : diviser les données, exécution d'un même programme par N processeurs, fusionner les sous résultats,
- de parallélisme de tâches : attribuer à N processeurs des traitements différents ou des traitements semblables (sur des données variables),
- de parallélisme de flux : mettre en pipeline un ensemble de traitements.

Une imbrication de ces différentes formes de parallélisme peut être réalisée. De plus, le concepteur fait appel à un modèle

de programmation parallèle, qui dans le cas d'une structure MIMD-DM, peut être par exemple basé sur le modèle de Processus Séquentiels Communiquants (CSP) [5] en utilisant une bibliothèque de communication par passage de message comme MPI [6]. Le passage "à la main" d'une version séquentielle à une version parallèle ne peut être réalisé que par des spécialistes et l'exploration de solutions est longue et difficile.

2.2 Qu'est ce que l'exploration d'architecture ?

Lors de l'implantation d'un algorithme sur une structure matérielle, l'exploration d'architecture se fait par l'intermédiaire d'environnement de prototypage rapide qui classiquement en codesign revient d'une part à décrire en langage d'abstraction élevé l'application et d'autre part d'évaluer de la manière la plus exhaustive possible l'ensemble des solutions de partitionnement HW/SW. Dans notre approche architecturale basée sur un réseau homogène de processeurs communicants, l'exploration d'architecture doit se faire relativement aux choix suivants :

- caractéristiques du softcore : taille pipeline, datapath, type et nombre d'ALU et FPU, mémoire cache, ...
- mémoire : type, taille, ...
- lien de communication : FIFO simple, routeur, protocole de communication, structure DMA, ...
- topologie fixe de communication : anneau, grille, tore, hypercube, ...
- interface avec les flots d'entrées/sorties du capteur,

et ceci en fonction des caractéristiques du programme parallèle décrites dans la section 2.1. L'ensemble de ces choix se fait grâce à l'utilisation d'une bibliothèque de blocs IP et à un outil de génération automatique du réseau vers l'outil de synthèse SoPC. Les performances à atteindre sont classiques d'un problème d'Adéquation Algorithme Architecture (AAA) et concerne l'accélération obtenue en fonction du nombre de processeurs tout en satisfaisant les contraintes matérielles et énergétiques.

3 Méthodologie proposée

3.1 Sur l'aspect de la parallélisation

Comme évoqué précédemment dans la problématique, un point-clé réside dans l'obtention du schéma de parallélisation de l'algorithme. Nous proposons, pour aborder ce problème, l'utilisation de l'outil d'aide à la parallélisation QUAFF, une bibliothèque C de développement d'applications parallèles à base de squelettes algorithmiques, développée au sein de notre laboratoire [8]. Cet outil permet le prototypage rapide de différentes solutions parallèles. Un squelette algorithmique est défini comme un schéma parallèle générique, paramètré par une liste de fonctions qu'il est possible d'instancier et de composer. Les fonctions sont spécifiées par l'utilisateur et chaque squelette encapsule les communications qui lui sont propres. A titre d'exemple, nous donnons trois types de squelettes : le SCM (Split Compute and Merge) qui correspond à un algorithme régulier comportant un parallélisme de données, le FARM (la ferme de processeurs, où un processeur maître distribue les données ou les traitements à ses processeurs esclaves et collecte les résultats) permet de traiter des algorithmes irréguliers, et le PIPE (pipeline) qui correspond à un parallélisme de flux. L'outil Quaff génère un code parallèle en incluant les fonctions de communications induites par la parallélisation du traitement, suivant le type de squelette spécifié par l'utilisateur. Les fonctions de communications que Quaff intègre dans le code C sont basées sur la bibliothèque MPI, qui peut être remplacée par une bibliothèque contenant les fonctions de communications propres à notre architecture.

3.2 Sur l'aspect de l'exploration d'architecture

Afin de répondre à la problématique de l'exploration d'architecture, nous devons permettre différentes spécifications du réseau de processeurs et assurer que leur implantation soit rapide. Nous avons donc développé plusieurs blocs IP de communication pour que les ressources - en terme de surface occupée par le réseau de communication - puissent être adaptées aux performances exigées par l'application visée. Nous avons développé deux blocs IP : un DMA (Direct Access Memory) et un Routeur de communication. Ceux-ci permettent l'instanciation de divers types de liens de communication, par exemple du plus simple au plus évolué : lien par FIFO, lien FIFO + DMA, lien ROUTEUR + DMA. Le nombre de liens est dépendant de la topologie et/ou du nombre de processeurs. Le choix du type de communication est important car il influe directement sur le nombre de processeurs pouvant être instancié au final sur un SoPC. Nous avons également conçu deux IP relatifs à l'acquisition du flot de pixels dans les mémoires processeur selon une approche double mémoire tampon ou une approche qui consiste à voir le flot pixel comme un lien de communication. Le processeur actuellement utilisé est le microBlaze Xilinx partiellement paramétrable. Le softcore Openfire [9] qui est un clone du précedent mais dont le code source est disponible a été testé pour donner des résultats similaires.

La construction du réseau de processeurs complet se résume à l'instanciation du nombre de softcore souhaité avec leur mémoire associée, et à effectuer les liaisons via l'IP de communication retenue. C'est lors de cette étape que la topologie du réseau est spécifiée. A chaque noeud est associé une adresse distincte et une stratégie de routage est choisie relativement à cette topologie et cet adressage. Dans ce but nous avons développé l'outil CubeGen. Il s'agit d'un script qui permet d'effectuer la spécification du système à 3 niveaux. Le premier au niveau de l'architecture réseau où sont définis la topologie et les règles d'adressage de chaque noeud. Un second niveau composant où l'on choisi les différentes IP. Puis le niveau micro-architecture, où les paramètres internes des composants sont spécifiés. Cu-

beGen intègre ces 3 niveaux de spécification et génère alors les fichiers d'entrée de l'environnement de développement EDK. Dans le principal fichier d'entrée (fichier ".mhs") se trouve les appels aux différentes IP retenues, et les informations liées aux connexions des composants. En utilisant l'outil CubeGen, on évite l'écriture manuelle de ce fichier ".mhs" qui avec la complexité du système devient source d'erreurs. La conception se résume alors à la spécification de quelques paramètres. Comme aucune étape d'implantation matérielle manuelle n'est nécessaire celà augmente la fiabilité de la méthode.

3.3 Liens entre les deux étapes

Les différentes étapes de notre méthode sont illustrées dans la figure 1. Les deux étapes principales sont celles situées au début du flot de conception : la parallélisation et la génération de l'architecture matérielle. L'idée principale de notre méthodologie repose sur l'extraction automatique de paramètres lors de la génération automatique du code parallèle, dans le but de paramétrer le réseau de processeurs afin de guider le choix des IP à utiliser pour générer son architecture. Ces paramètres fournissent à l'utilisateur un point de départ convenable pour le guider vers la configuration la mieux adaptée.

Par exemple : recenser les instructions utilisées, permet d'extraire des informations sur le format et la précision des calculs (opérations logiques, calculs en nombres entiers, ou en nombres à virgule flottante) ce qui donne des critères pertinents pour choisir une architecture de processeur. L'analyse du code et le profilage de l'application permettent d'évaluer les effets du conditionnement et de la localité des données ainsi que la consommation en ressource mémoire pour chaque processeur. De la même manière, l'évaluation des besoins en communications peut servir de critère pour le choix des dispositifs de communications, aussi bien sur la nature matérielle des liens que sur la topologie du réseau (hypercube, anneau...).

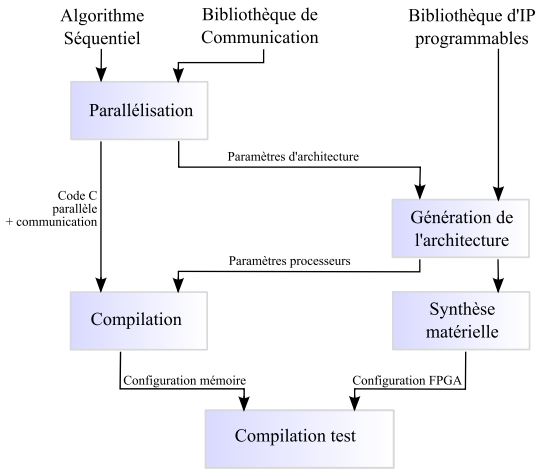

FIG. 1 – Flot de conception

Il apparait à la suite de ces deux étapes les étapes classiques d'un cycle de développement de SoPC. Après la compilation du code C parallèle, les fichiers de configuration mémoire de chaque processeur sont obtenus et pourrons être placé. L'étape de Synthèse matérielle qui comprend : la synthèse RTL, le placement technologique et le placement routage, donne les fichiers de configuration de la cible FPGA pour l'implantation de l'architecture.
En fonction des résultats obtenus il est possible de revenir sur les choix initiaux des paramètres d'architecture, afin de converger vers une architecture plus adéquate aux contraintes de l'application.

4 Résultats actuels

En ce qui concerne l'exploration de l'espace d'architecture, nous avons étudié diverses configurations de processeurs et de liens de communications (point à point type FIFO ou point à point avec DMA), et l'influence que ces choix pouvaient avoir sur la consommation de ressources SoPC. Les tableaux 1 et 2 présentent les ressources utilisées pour le cas de réseau de 16 ou 32 processeurs en fonction des différents liens de communication. Plus de détails sur cet aspect peuvent être retrouvés dans [2].

DEPTH	NB μP	SLICE	FF	LUT4	BRAM
1	32	29%	8%	22%	76%
64	32	60%	7%	55%	76%
128	32	113%	37%	86%	76%
256	32	152%	9%	150%	76%
1	16	13%	3%	10%	38%
64	16	26%	4%	24%	38%
128	16	61%	15%	36%	38%
256	16	64%	4%	61%	38%

TAB. 1 – Ressources utilisées pour différentes tailles de FSL sur cible FPGA XC4VLX200.

TYPE	NB μP	SLICE	FF	LUT4	BRAM
HD	32	93%	40%	71%	85%
FD	32	108%	50%	81%	114%
HD	16	38%	16%	30%	38%
FD	16	44%	20%	34%	48%

TAB. 2 – Ressources utilisées pour les liaisons DMA Half Duplex et Full Duplex sur cible FPGA XC4VLX200.

Une solution plus complexe de communication, basée autour d'un routeur de paquets, a aussi été mise en place. Ce routeur hardware offre de meilleur performances de communications, puisqu'il permet de décharger le processeur de la gestion des communications. Un réseau en hypercube de 8 processeurs avec 8 routeurs a été implanté et testé sur SoPC (XC4VLX60). Ce routeur hardware utilise quasiement autant de ressources logique qu'un softcore de type microBlaze, mais les gains en

performances de communication sont importants. Des détails sur l'architecture et les apports de ce routeur sont publiés dans [3].

En ce qui concerne l'implantation d'un algorithme de TSI sur réseau de processeurs embarqué, nous avons étudié le cas de la stabilisation d'images. Cet algorithme a été parallélisé et porté sur un réseau de processeurs embarqués sur SoPC. Comme cet algorithme ne présente pas un fort besoin en communication, nous avons fait le choix d'utiliser de simple liens de communication par liaisons FSL. Nous avons étudié les performances de l'application en fonction du nombre de processeurs (de 1 à 64). Comme le montre le tableaux 3, on peut remarquer que l'accélération est quasi linéaire et que pour un réseau de 8 procsesseurs, le système fonctionne à temps réel. De plus amples détails sur cette implémentation se trouvent dans [4].

TAB. 3 – 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**

Le flot de conception présenté en figure 1 a été validé. Cependant, l'obtention des paramètres de l'architecture matérielle à partir du profil de l'application parallèle n'est pas encore un processus automatisé. D'autre part, les outils de parallélisation se basent actuellement sur une librairie de communication standard (MPI) pour la génération du code parallèle. Ils peuvent cibler des architectures de type cluster de stations de travail, mais aussi des architectures embarquées (processeur Cell [7]). Une étape de portage supplémentaire est cependant nécessaire pour l'implantation de l'algorithme parallèle sur cible SoPC Final.

5 Conclusion-Perspectives

Nous proposons une approche pour l'implantation de traitement d'images sur des systèmes électroniques embarqués. La méthode se base sur un approche en deux étapes basées sur des outils d'aide à la parallélisation, et de génération automatique d'une architecture matérielle, l'implémentation se faisant à partir d'un ensemble de blocs IP paramétrables. Les outils utilisés pour mettre en oeuvre notre approche permettent l'implantation rapide de prototypes d'architectures comportant un nombre élevé de processeurs et leur test sur Cible SoPC Xilinx. L'approche décrite n'est pas un codesign classique, le partitionnement est remplacé par une étape de parallélisation. Ensuite, au prix de restrictions dans les choix architecturaux, elle offre à terme d'être une approche fortement automatisée, ce qui permet des temps de conception réduits et une certaine fiabilité des processus de conception. De plus, cette approche nécessite des moyens limités en terme de ressources humaines (équipe réduite) et non un ensemble d'experts dans tous les domaines touchés par l'application (traitement d'image, traitement parallèle, conception électronique etc ...). Pour pouvoir augmenter le nombre de paramètres au niveau du softcore, nous prospectons un ensemble d'"open softcores" ainsi que l'environnement de type "Language Instruction Set Architecture" [1] qui offre la possibilité de générer son propre coeur de processeur.

Références

[1] O. Schliebusch, A. Hoffmann, A. Nohl, G. Braun et H. Meyr. *Architecture Implementation Using the Machine Description Language LISA* VLSID 2002. 15th International Conference on VLSI Design, IEEE Computer Society, 2002.

[2] J. P. Derutin, L.Damez, A. Desportes, et J.L. Lazaro Galilea. *Design of a Scalable Network of Communicating Soft Processors on FPGA*. CAMPS 2006. Computer Architecture for Machine Perception and Sensing, IEEE Computer Society, 2006.

[3] L. Damez et J.P. Dérutin. *Fast prototyping of complex Signal and Image Processing applications on SoC using homogenous network of communicating processors* PACT 2007. Parallel Architectures and Compilation Techniques, IEEE Computer Society, 2007.

[4] J.P. Dérutin, L. Damez, et A. Landrault. *Embedding of a Real Time Image Stabilization Algorithm on SoPC Platform, a Chip Multi-processor Approach* ACIVS 2008. Avanced Concepts for Intelligent Vision Systems, Springer, 2008.

[5] C.A.R Hoare, *Communicating Sequential Processes* Communications of the ACM, ACM, 1978.

[6] the Message Passing Interface Forum, *MPI : A Message-Passing Interface Standard*

[7] J. Falcou, T. Saidani, L. Lacassagne et D. Etiemble. *Programmation par squelettes algorithmiques pour le processeur CELL* RenPar'18 - SympA'2008 - CFSE'6 - Actes de Fribourg 2008.

[8] J. Falcou, J. Serot, T. Chateau, J.T. Lapresté *QUAFF : Efficient C++ Design for Parallel Skeletons* Parallel Computing, Volume 32, Issues 7-8, Septembre 2006, Pages 604-615.

[9] S. Craven, C.Patterson, and P. Athanas *Configurable Soft Processor Arrays Using the OpenFire Processor* 2005 MAPLD Int. Conf. , 7-9 september, Washington DC Bradley Department of Electrical and Computer Engineering Virginia Polytechnic Institute and State University, Blacksburg, VA 24061