Disydent : un environnement pour la conception de systèmes numériques synchrones

Ivan Augé, Frédéric Pétrot†, Richard Buchmann, François Donnet, Pascal Gomez, et Étienne Faure
Département Architecture des Systèmes Intégrés et Micro-électronique
LIP6, Université Paris VI, Paris, France

***Résumé*— Ce papier présente l'environnement DYSIDENT destiné a la conception de systèmes intégrés numériques à base de plate-forme. Nous définissons un problème de conception sur plate-forme comme un triplet *(système, nouvelle application, contraintes)*, ou le *système* est à la fois un système d'exploitation minimal et un gabarit architectural général dans lequel on peut faire varier par exemple le nombre de processeurs et dans lequel on peut ajouter des coprocesseurs dédiés.**

Le concepteur modélise en C son application sous la forme d'un réseau de processus de Kahn, en utilisant la bibliothèque de fonctions DPN. La validation fonctionnel a lieu par simple exécution sur la machine hôte. Les caractéristiques temporelles sont obtenues par simulation de la plate-forme exécutant l'application au niveau cycle, grâce à l'outil CASS. En fonction des résultats obtenus, le concepteur sélectionne des ensembles de processus candidats à une implantation matérielle. Les processus sont ensuite synthétisés grâce à l'outil de synthèse de haut niveau UGH, qui non content de produire une description RTL synthétisable fourni également un modèle CASS précis au cycle.

Les outils de Disydent (DPN, CASS, UGH) sont intimement liés, ce qui facilite grandement le passage entre les différentes étapes de la conception.

I. INTRODUCTION

La notion de *codesign* est utilisée dans la littérature dès que du matériel et du logiciel sont utilisés conjointement. Cette notion recouvre en fait des domaines très différents. Certains auteurs font beaucoup de matériel et un minimum de logiciels, et *vice-versa*. Par exemple [1], [2], [3] examinent un nid de boucle et génèrent un coprocesseur à architecture systolique destiné uniquement à l'exécution de ce nid. Le logiciel est réduit à l'échange de données entre le processeur et son coprocesseur. À l'autre bout du spectre, [4], [5], [6] étendent des processeurs existant avec quelques accélérateurs matériel et se focalisent sur la détection de code pouvant être exécuté plus efficacement grâce à ces accélérateurs.

De même, la cible finale de l'implantation peut prendre des formes très variées. Ce peut être un ordinateur pilotant une carte programmable (à base de FPGA), une carte contenant un processeur de traitement du signal (DSP) avec des coprocesseurs spécialisés, ou encore un système soit sur puce, soit sur une carte *ad-hoc*. Dans tous ces cas, des techniques de *codesign* employées sont assez différentes. [5], [6] utilisent une approche VLIW. L'idée est d'extraire le parallélisme au niveau instruction (ILP) pour générer une machine de type instructions multiples pour donnée unique (MISD dans la classification de Flynn [7]). [1], [2] font usage de projection spatiale et temporelle des nids de boucles pour produite une machine à instruction unique sur données multiples (SIMD). [8], [9] extraient le parallélisme à gros grain pour produire une machine à instructions multiples sur données multiple (MIMD). Concernant l'implantation effective, elle peut être faite à partir de rien, ou sur une plate-forme contenant déjà des composants matériel et logiciel.

Cet article présente Disydent, un environnement dédié à la conception de systèmes sur puce de type MIMD sur plate-forme préexistante. Nous nous concentrons ici sur l'utilisation de cet environnement par un concepteur de système après que Disydent ait été configuré pour une plate-forme donnée.

†Auteur à contacter : Frederic.Petrot@lip6.fr

II. PRINCIPES DE DISYDENT

A. Définition de la conception sur plate-forme

Le problème que nous cherchons à résoudre est l'extension d'un système particulier (tel un téléphone mobile ou un assistant numérique) en y intégrant une nouvelle application. Nous pouvons ainsi définir le problème de la conception sur plate-forme comme un triplet *(système, nouvelle application, contraintes)* dans lequel le système est déjà défini mais l'application n'est pas supportée par le système. Nous entendons par *système* du matériel et du logiciel permettant l'exécution d'autres applications. Cependant, nous nous réservons la possibilité de l'améliorer en ajoutant des accélérateurs matériels spécifiques à une application donnée.

Intégrer de nouvelles fonctionnalités à un système sur puce impose généralement que :

1) les accélérateurs matériels fonctionnent à la fréquence du système ;
2) les accélérateurs matériels utilisent la même technologie de fabrication ;

3) des pilotes logiciels et des composants matériels spécifiques sont utilisés pour permettre les communications entre matériel et logiciel ;
4) des techniques de validations fonctionnelles et temporelles puissent être mises en œuvre pour s'assurer de la pertinence de l'implantation.

B. Entrées

Afin d'effectuer l'implantation de son triplet *(système, nouvelle application, contraintes)*, l'utilisateur doit fournir à l'environnement les entrées définies ci-dessous :

1) un modèle exécutable de son application décrite dans une forme restreinte des réseaux de processus de Kahn (KPN) [10]. Un KPN est un ensemble de processus communicant à travers des files de taille infinie. Les files possèdent un unique producteur et un unique consommateur. La lecture est bloquant si la file est vide. La restriction de Parks [11], que nous utilisons, consiste à fixer la taille des files. Dans ce cas la primitive d'écriture est bloquante lorsque la file est pleine. Le langage de description utilisé est le langage C ;
2) un système étendu, constitué de matériel et d'un système d'exploitation. Ce système étendu est construit en ajoutant des coprocesseurs au système **asim0** que nous utilisons comme exemple dans cet article (voir la figure 1). La partie matérielle est composée d'un nombre $p \geq 1$ de processeurs, de bus, d'interfaces de communication de type files, de contrôleurs d'interruptions, de contrôleurs d'accès direct à la mémoire (DMA), ... Elle doit également contenir les coprocesseurs implantant les processus de Kahn dont il a été décidé qu'ils seraient matériels.
   Le système d'exploitation est un micro-noyau qui implante la spécification des tâches POSIX[12]. Il propose également une couche de communication par files permettant de réaliser les communications telles que définies par les KPN avec des files dont l'une ou les deux extrémités peuvent être en matériel [13].

Quelques commentaires sur ces choix : I) le premier point des spécifications peut sembler restrictifs vis-à-vis des moyens de communication, car les files dans les KPN excluent les communications par mémoire partagée, par sémaphore et par interruption. Ceci est en partie faux, car : 1- de tels communications sont envisageable entre les processus logiciels ; 2- le concepteur est libre de ce qu'il fait dans un bloc matériel ; 3- la sémantique des files bloquantes est implémentée grâce à ces mécanismes bas niveaux ; 4- les communications peuvent même être implantées à l'aide de fonctions complexes, tel les engins de DMA.

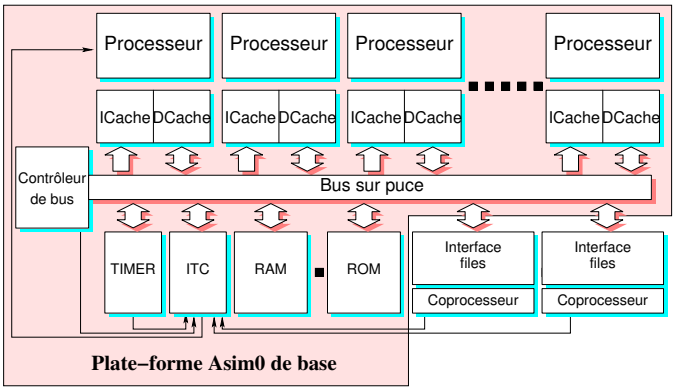

Fig. 1. Le matériel asim0 étendu.

De notre point de vue, ce « moule » n'est pas une contrainte mais un moyen de concevoir des systèmes en se reposant sur des composants sûrs et optimisés. L'avantage d'une telle approche est la rapidité, car elle permet d'éviter l'invention de nouvelles manières de communiquer pour lesquels il faudra dans tous les cas du matériel approprié.

II) le second point restreint le système à des instances de la plateforme **asim0**. C'est certes le cas, mais la plate-forme fait partie de la spécification. De fait, concevoir une plate-forme est un travail fort différent de celui consistant à implanter une application sur une plate-forme. Ces deux aspects sont généralement réalisé par des personnes différentes.

Dans un environnement comme SystemC, qui vise à la fois le développement de la plate-forme et de l'application, nous aurions : un expert qui fourni la bibliothèque des KPN, un autre expert qui fourni les modèles des composants matériels de la plate-forme, et un troisième expert fourni le micro-noyau. Le travail de l'ingénieur d'application est alors plus difficile que le travail individuel de chaque expert, car il nécessite une bonne partie de leurs connaissances respectives.

C. Migration d'un processus du logiciel vers le matériel

Le réseau de Kahn peut être exécuté en tant que tel sur une machine hôte pour vérifier sa fonctionnalité. Il peut aussi être simulé au niveau cycle sans modifications sur une plate-forme **asim0** étendue sans modification pour vérifier son fonctionnement et obtenir ses performances. Pour exécuter un réseau de Kahn sous forme de coprocesseur, Disydent propose deux méthodes :

– la migration exploratoire. La tâche POSIX réalisant le processus de Kahn peut être directement connectée à la plate-forme par l'intermédiaire d'un convertisseur du monde de l'hôte vers le monde du simulateur cycle. Dans ce cas, la simulation de la tâche ne prend pas de temps entre deux opérations d'entrées/sorties.
– la migration réelle. La description de la tâche est réécrite

en vue de la synthèse d'architecture, en utilisant un C modifié (sans pointeurs, en spécifiant des tailles en bits sur les variables, etc). Cette nouvelle description peut être substituée à la précédente à tous les niveaux, notamment pour en vérifier l'exactitude. L'outil de synthèse de Disydent produit alors un modèle de simulation au cycle qui peut être connecté directement aux files d'attentes matérielles. Dans ce cas, l'exécution sur le simulateur fournit des temps précis au cycle entre les entrées sorties.

La migration du logiciel vers le matériel est relativement aisée car la seule manière de communiquer est à travers des files. L'outil de synthèse possède les primitives de communication par file et la plate-forme **asim0** fournit les modules de communication par files ainsi que leurs pilotes logiciels. Le concepteur doit donc simplement :

au niveau logiciel

supprimer le lancement de la tâche logiciel et configurer les extrémités des files de ce processus comme étant matérielles ;

au niveau matériel

créer la plate-forme étendue. Elle est constituée de la plate-forme **asim0** à laquelle on ajoute le module de communication par file et le modèle généré par la synthèse.

D. Outils

Il y a dans Disydent 4 outils distincts permettant d'aller de la spécification fonctionnelle à l'implantation effective. **DPN** est une bibliothèque C/C++ qui fourni les communications par files du modèle de Kahn, en utilisant les tâches POSIX comme support d'exécution ;

CASS est un simulateur de modèles précis au bit et au cycle. Les modèles de simulation sont écrits en C et des techniques d'ordonnancement statique sont utilisées pour atteindre les meilleurs vitesses de simulation possibles [14].

ASIM0 est une plate-forme dont le matériel est décrit en VHDL RTL synthétisable et dont le logiciel est un micro-noyau décrit en C. Les réseaux de Kahn sont supporté par le matériel et le logiciel indépendemment de la nature des tâches. De plus, chaque composant possède un modèle de simulation CASS.

UGH est un outil de synthèse de haut niveau qui prend en entrée un processus C, une esquisse du chemin de données et une fréquence de fonctionnement désirée, et produit un modèle VHDL synthétisable et un modèle de simulation CASS[15].

III. Flot de conception

A. Exemple d'un décodeur Multi-JPEG

Nous illustrons l'approche de conception préconisé par Disydent avec le triplet *mjpeg* définit comme suit :

mjpeg = (asim0 avec des périphériques d'entrée et de sortie,
décodeur Multi-JPEG,
fréquence de 33 MHz pour 20 images/sec)

Le périphérique d'entrée est un générateur de traffic noté TG, et le périphérique de sortie est un convertisseur vidéo noté RAMDAC.

Cette application lit un flux d'images JPEG 64x64 et produit un flux de *pixels* dans l'ordre des lignes pour affichage. Comme on peut le voir sur la figure2, le décodage d'une image nécessite 6 étapes.

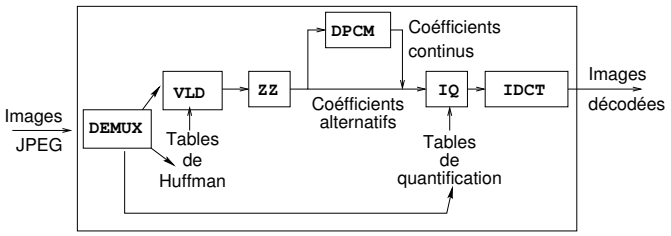

Fig. 2. Principe du décodage de flux Multi-JPEG

1) DEMUX analyse le flux fournit par le périphérique d'entrée pour en extraire la taille de l'image, les tables de Huffman et les tables de quantification. Ces informations sont usuellement stockées dans des variables globales. Les données de l'image compressée sont lues par paquet et rangées dans un tampon.
2) le décodeur de Huffman, VLD, décompresse ce tampon et met le résultat dans un deuxième tampon.
3) ZZ réorganise le tampon suivant l'ordre zigzag et produit son résultat dans un troisième tampon.
4) IQ effectue la quantification inverse du tampon précédent pour le mettre dans un nouveau tampon.
5) IDCT exécute une transformé discrète inverse en cosinus du tampon fournit par IQ dans un nouveau tampon.
6) LIBU stocke les blocs (8x8) issus de IQ dans un sixième tampon dont la largeur en bit correspond à la taille de l'image. Une fois les lignes disponibles, elles sont émise vers le périphérique de sortie.

La spécification initiale est un programme C séquentiel qui lit les images comprimées d'un fichier et qui écrit les *pixels* résultant dans un fichier.

B. Analyse des performances du code

La première implantation consiste à porter la spécification séquentielle sur la plate-forme **asim0**. Il faut en conséquence modifier les fonctions d'entrée/sortie pour les adapter aux composant de la plate-forme cible.

L'exécution de l'application produit des informations de latence et de débit en cycle. Si ces résultats entrent dans les contraintes spécifiées, alors la conception s'achève. Dans le cas contraire, des informations temporelles plus précises sont nécessaires. Un profilage est effectué en simulant de nouveau et en notant le nombre de cycles passés dans chaque fonction. Une description sous forme de KPN est alors nécessaire.

C. Description sous forme de KPN

En utilisant les informations de profilage, le concepteur doit extraire le parallélisme à gros grain et modéliser l'application sous forme de KPN. Pour notre exemple, le réseau est présenté

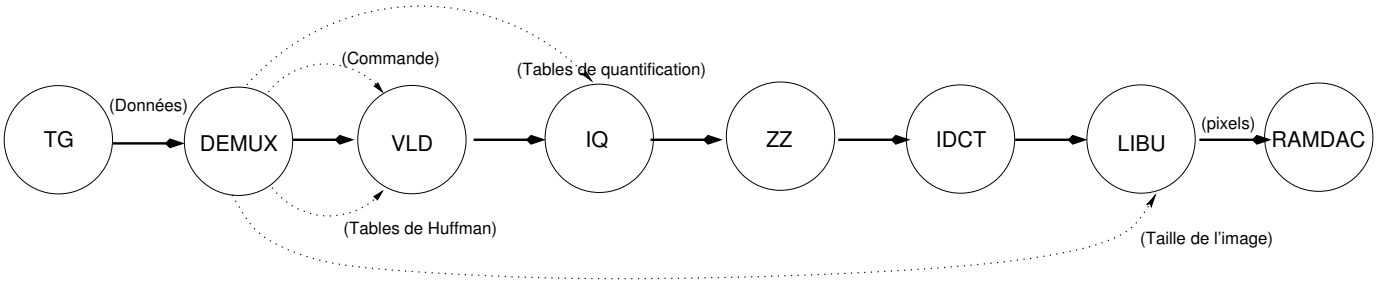

Fig. 3. Un réseau de Kahn possible pour l'application Multi-JPEG

sur la figure 3. Le décodage JPEG ne peut être parallélisé, car le décodage de Huffman est intrinsèquement séquentiel, mais puisque nous avons un flux d'images, nous pouvons pipe-liner les traitements (nous avons choisi de le faire au niveau du macro-bloc JPEG).

Les arcs en gras représentent le flux de décompression, et les arcs en pointillés représentent les paramètres de configuration, qui furent des variables globales dans la configuration initiale.

L'application est réécrite en utilisant les primitives DPN. Elle est compilée pour la machine hôte, puis exécutée pour s'assurer de sa fonctionnalité.

Ensuite, l'application est compilée pour la plate-forme cible, puis exécutée avec un nombre croissant de processeurs jusqu'à ce que le temps d'exécution stagne. Si les contraintes de temps, surface, consommation sont satisfaites, alors la conception est finie. Sinon, il faut faire des accélérateurs matériels pour certains processus de Kahn.

D. Adaptation de la plate-forme

La conception de matériel a un fort coût en main d'œuvre, donc avant de se lancer dans un nouveau développement, il faut d'une part s'assurer qu'il sera réellement utile et d'autre part définir les performances que l'on en attend. Ce deuxième point est crucial, car atteindre, sur une fonction, une accélération d'un facteur 5 ou d'un facteur 100 à un coût fort différent en effort de conception et en surface/consommation.

Le problème se ramène à trouver un ensemble de processus qui sont de bons candidats à l'implantation matérielle. Pour résoudre ce problème, l'utilisateur doit sélectionner un sous-ensemble des processus de Kahn à migrer en matériel, en se basant sur les informations de profilage et à son intuition.

La plate-forme étendue sur laquelle l'application s'exécute est simulée. Deux cas se présentent alors : I) les temps semblent donner assez de marge pour réaliser un matériel qui pourra respecter les contraintes. Cette décision ne peut être prise que par un concepteur de circuits. II) les contraintes ne sont pas respectées ou le sont avec très peu de marge. Dans ce cas un nouvel ensemble de processus doit être choisi. Si aucun groupe ne fait l'affaire, la parallélisation à gros grain doit être remise en cause. Si aucune modélisation en KPN ne rentre dans les contraintes, alors le problème posé par le triplet *mjpeg* est sans solution pour le concepteur.

E. Synthèse matérielle des processus

Les étapes précédentes ont permis de déterminer un ensemble de processus à réaliser en matériel qui doivent être ajoutés à la plate-forme. Pour créer et valider cette nouvelle plate-forme, le concepteur suit dans les grandes lignes la méthode suivante :

1) **Adaptation des processus aux contraintes de UGH**. Chaque processus doit être modifié pour s'adapter aux contraintes de UGH. Pour cela, il faut définir une esquisse du chemin de données, c.-à-d. fondamentalement les registres du circuit, et faire en sorte que les variables du comportement C portent le nom de ces registres, à moins qu'elles puissent être absorbées par les instructions qui en font usage (*inlining*) ;
2) **Validation fonctionnelle des processus UGH**. La description C pour UGH est compatible avec les primitives DPN, ainsi elle peut être simulée à la place de la description initiale, à la fois sur la machine hôte et sur la plate-forme cible. Ceci permet de s'assurer du comportement avant d'effectuer la synthèse, et indépendamment de l'esquisse du chemin de données.
3) **Validation temporelle**. La synthèse du processus est effectuée pour la fréquence de la plate-forme, afin d'obtenir dans un premier temps un modèle CASS précis au cycle. La plate-forme est étendue par l'ajout des modules ainsi synthétisés et l'application est lancée pour obtenir les temps exacts.
4) **Validation des coûts**. Si les contraintes temporelles sont respectées, UGH est relancé pour obtenir la description VHDL RTL. Cette description est ensuite synthétisée par les outils classique d'optimisation booléenne et de projection structurelle, afin d'obtenir la surface et la puissance consommée. Si les résultats satisfont les contraintes, alors la conception est achevée. Dans le cas contraire, le concepteur doit améliorer soit la micro-

architecture du processus matériel, ou encore choisir d'autres processus à réaliser en matériel, ou encore remettre en cause sa description en KPN.

Le point négatif de cette approche est la nécessité de réécrire le processus pour entrer dans le moule imposé par UGH. Il y a donc deux descriptions du même objet. Bien que la version pour UGH puisse également être utilisée à la place de la description initiale, son exécution en logiciel est généralement plus longue. En réalité, les deux versions sont vraiment utiles, car les bons algorithmes pour le matériel sont très différents des bons algorithmes pour le logiciel. En effet, la description pour le matériel met en avant le parallélisme à grain fin à travers une description séquentielle qui rend le code plus complexe, et des calculs sur un nombre de bit précis sont mis en œuvre en lieu et place des opérations sur les types du C, afin de ne pas inclure du matériel inutile.

IV. Mise en œuvre de la méthode sur un exemple

Les expériences sont effectuées sur notre triplet *mjpeg*.

Nous avons simulé le système **asim0** avec l'application totalement logicielle pour un nombre de processeurs croissant. Ces simulations donnent le nombre de cycles nécessaire au décodage de 20 images et le nombre de cycles passé dans chaque processus. La table I donne le pourcentage du temps passé dans chaque tâche pour une exécution avec notre séquence de test sur un processeur. La courbe logiciel de la figure 4 est le tracé de ces nombres de cycles ramenés en images par seconde pour une fréquence de fonctionnement de 33 MHz.

Tab. I
Profilage de l'application JPEG

tâche	% du temps
DEMUX	1,49%
VLD	17,17%
IQ	11,77%
ZZ	12,14%
IDCT	42,86%
LIBU	10,19%
commutation	4,37%
initialisation	0,03%
total	100%

l'*initialisation* et la *commutation* correspondent au temps passé dans le noyau

L'analyse de ces résultats montre que des accélérateurs matériels sont nécessaires pour atteindre la contrainte de 20 images par seconde. Le processus IDCT est le meilleur candidat à la migration. La courbe IDCT exploratoire de la figure 4 représente la simulation d'un système **asim0** dans lequel l'IDCT à subit une migration exploratoire.

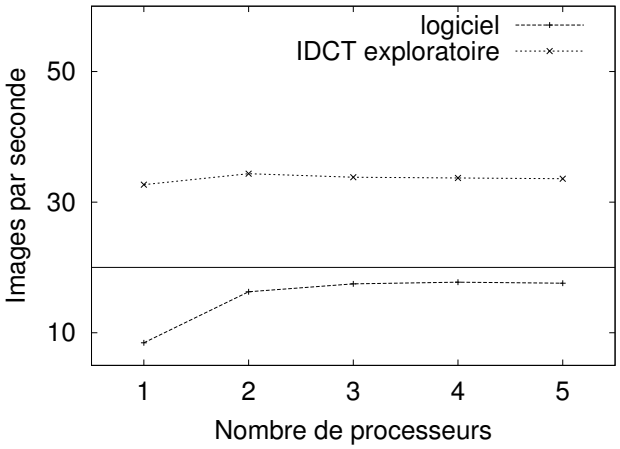

Fig. 4. Performances pour une implantation purement logicielle et un coprocesseur IDCT

Bien que prometteuse, la complexité intrinsèque de l'algorithme est telle que la migration d'un autre processus est probablement nécessaire.

Nous avons réalisé des modèles UGH pour l'IDCT, le VLD et l'IQ, et les avons synthétisés afin d'obtenir des modèles de simulation CASS. Nous avons exécuté ces simulations pour 3 ensembles : (IDCT), (IDCT, VLD), (IDCT, IQ). Les débits en images par seconde sont tracés sur le figure 5. Les meilleurs candidats sont (IDCT, IQ) avec un processeur, (IDCT, VLD) avec un processeur et (IDCT) avec 2 processeurs. Le choix final est donnée par le coût en surface et/ou puissance consommée. La surface après synthèse RTL est donnée dans la table II.

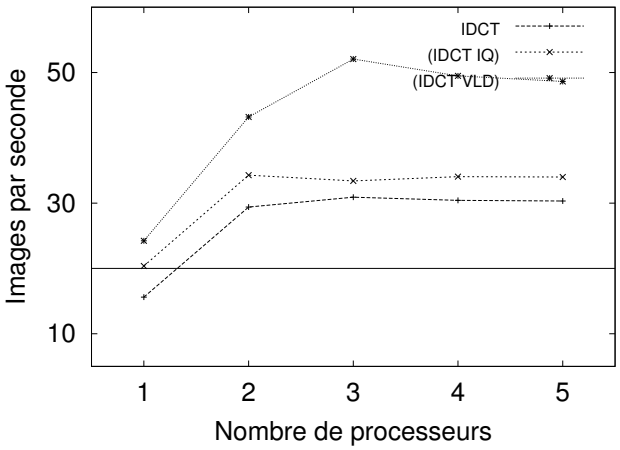

Fig. 5. Performances pour différents ensemble de processus matériels

L'ensemble (IDCT, IQ) est la meilleure solution car IQ est 20 fois plus petit que le VLD, et également bien plus petit qu'un CPU (MIPS R3000) avec ses caches.

Tab. II
Surface des processus matériels en mm^2.

IDCT	VLD	IQ
10.9	**13.2**	**1.2**

V. Conclusion

La configuration de Disydent pour une nouvelle plate-forme est une tâche ardue. Elle exige *a)* la définition d'un système de base (composants matériels, types de communication supporté, micro-noyau, pilotes du matériel, ...) ; *b)* la conception au niveau RTL synthétisable des composants matériels la constituant et le développement de leurs pilotes ; *c)* le développement de modèles de simulation CASS pour la simulation cycle.

Cependant, une fois ce travail achevé, l'intégrateur d'application, c.-à-d. l'utilisateur de Disydent, doit uniquement étendre la plate-forme et optimiser la projection de son application sur l'existant.

Les principaux avantages de Disydent sont les suivants :

- la spécification exécutable étant un programme C utilisant les primitives des réseaux de Kahn, la validation fonctionnelle est effectuée à très hautes performances sur un ordinateur classique. Par exemple le décodage de films mpeg2 en temps réel est possible pour des petites tailles d'image. Ceci doit être comparé aux performances des simulateurs de matériels classiques qui sont extrêmement lents ;
- la validation temporelle, grâce au simulateur au niveau cycle, est plus rapide d'un ou deux ordres de grandeur par rapport aux simulateurs événementiels classiques (VHDL ou SystemC) et plus lents de quelques ordres de grandeurs que la simulation sur l'hôte (tant que le nombre de composants à simuler reste raisonnable) ;
- la validation fonctionnelle au niveau KPN est suffisante pour garantir la fonctionnalité sur la plate-forme cible. Ceci est du à la propriété fondamentale des KPN qui garantit que la vitesse relative d'exécution des processus n'influe pas sur le comportement de l'application[10] ;
- les liens étroits entre les outils permet de développer une application comme le Multi-JPEG en quelques semaines.
  - la description du réseau de Kahn est exécutable sur l'ordinateur hôte et sur la plate-forme ;
  - un processus peut être synthétisé par l'outil de synthèse de Disydent. Celui-ci génère directement un modèle de simulation cycle, là ou d'autres outils de synthèse ne génèrent qu'une interconnexion de portes logiques. Dans ce cas, il faut recourir à des techniques de cosimulation qui sont très lentes, car pilotées par les évènements, et qui nécessitent de l'ingénierie pour les mettre au point ;
  - la migration exploratoire permet d'estimer au plus tôt les contraintes temporelles requises par un coprocesseur, en prenant en compte les délais de communication. Ceci aide à éviter de se lancer dans la conception de matériel inutilement.

L'environnement Disydent est distribué sous licence GPL et est disponible sur le web à l'URL `www-asim.lip6.fr/recherche/disydent`.

Références

[1] P. Quinton and V. van Dongen. The mapping of linear recurrence equations on regular arrays. *Journal of VLSI Signal Processing*, 1(2) :95–113, October 1989.

[2] J. Xue and C. Lengauer. The synthesis of control signals for one-dimensional systolic arrays. *Integration, the VLSI journal*, 14(1) :1–32, 1992.

[3] P.-Y. Calland and T. Risset. Precise tiling for uniform loop nests. In *International Conference on Application Specific Array Processors*, pages 330–337, 1995.

[4] G. Goossens, J. Van Praet, D. Lanneer, W. Geurts, and F. Thoen. *Programmable Chips in Consumer Electronics and Telecommuncations – Architectures and Design Technology*, pages 135–164. Kluwer Academic Publishers, 1996.

[5] S. Aditya, B. R. Rau, and V. Kathail. Automatic architectural synthesis of VLIW and EPIC processors. In *ISSS*, pages 107–113, 1999.

[6] P. Faraboschi, G. Brown, J. A. Fisher, G. Desoli, and F. Homewood. Lx : a technology platform for customizable VLIW embedded processing. In *The 27th Annual International Symposium on Computer architecture 2000*, pages 203–213, New York, NY, USA, 2000. ACM Press.

[7] M. Flynn. Very high-speed computing systems. *Proceedings of the IEEE*, 54 :1901–1909, December 1966.

[8] R. K. Gupta and G. De Michelli. Hardware-software cosynthesis for digital systems. *IEEE Design and Test of Computers*, 10(3) :29–41, September 1993.

[9] E.A. de Kock, G. Essink, W.J.M. Smits, P. van der Wolf, J.-Y. Brunel, W.M. Kruijtzer, P. Lieverse, and K.A. Vissers. YAPI : Application modeling for signal processing systems. In *37th Design Automation Conference*, pages 402–405, 2000.

[10] G. Kahn. The semantics of a simple language for parallel programming. In *Information Processing 74*, pages 471–475, Stockolm, Sweden, August 1974.

[11] T. M. Parks. *Bounded Scheduling of Process Networks*. PhD thesis, Electronics Research Laboratory, Berkeley, 1995.

[12] F. Pétrot, P. Gomez, and D. Hommais. Lightweight implementation of the posix threads api for an on-chip mips multiprocessor with vci interconnect. In Ahmed Amine Jerraya, Sungjoo Yoo, Diederik Verkest, and Norbert Wehn, editors, *Embedded Software for SoC*, part 1, chapter 3, pages 25–38. Kluwer Academic Publisher, November 2003.

[13] J-Y. Brunel, W. M. Kruijtzer, H. J. H. N. Kenter, F. Pétrot, L. Pasquier, E. A. de Kock, and W. J. M. Smits. Cosy communication ip's. In *37th Design Automation Conference*, pages Pages 406–409, Los Angeles, CA, June 2000.

[14] F. Pétrot, D. Hommais, and A. Greiner. Cycle precise core based hardware/software system simulation with predictable event propagation. In *Proceeding of the* 23rd *Euromicro Conference*, pages 182–187, Budapest, Hungary, September 1997. IEEE.

[15] I. Augé, R. K. Bawa, P. Guerrier, A. Greiner, L. Jacomme, and F. Pétrot. User guided high level synthesis. In *VLSI : Integrated Systems on Silicon*, Gramado, Brazil, August 1997. Chapman & Hall.