CODEF : un environnement d'exploration d'architecture de systèmes embarqués pour le traitement du signal temps réel

Luc BIANCO[1], Michel AUGUIN[1], Emmanuel GRESSET[2], Alain PEGATOQUET[1,2]

[1]I3S, Université de Nice Sophia Antipolis, CNRS
Les Algorithmes / Bâtiment Euclide B, 2000 route des Lucioles, BP 121, 06903 Sophia-Antipolis Cedex, France

[2]VLSI Technology
505 Route des Lucioles, Sophia Antipolis, 06560 Valbonne, France

bianco@i3s.unice.fr, auguin@i3s.unice.fr
Emmanuel.Gresset@vlsi.com, Alain.Pegatoquet@vlsi.com

Résumé –La complexité croissante des applications nécessite de nouvelles méthodes pour le prototypage rapide de systèmes temps réels orientés traitement du signal. L'approche proposée est basée sur un algorithme de partitionnement capable d'explorer automatiquement un espace de conception système. Les solutions construites respectent les contraintes temporelles de l'application et exploitent les mobilités des tâches pour optimiser la surface globale. L'algorithme de partitionnement est également interactif dans le but d'aider le concepteur à optimiser plus finement une solution, par exemple obtenue lors de l'exploration automatique.

Abstract– Due to the increasing complexity of real time signal processing applications, new rapid prototyping methods are required. Our approach is based on a partitioning algorithm that is able to perform an automatic system design space exploration. The provided solutions satisfy the time constraints of the application and take into account the mobility of the tasks to reduce the total area. The partitioning algorithm provides some user interactions in order to assist the designer to improve a solution, for example resulting of the automatic space exploration.

1. Introduction

Le prototypage système pour des applications embarquées de traitement du signal soumises à des contraintes temporelles est une des étapes primordiales du processus de conception car des choix essentiels y sont souvent réalisés. A partir des spécifications fonctionnelles de l'application cible, il est important de déterminer le plus tôt possible dans le processus de conception l'architecture d'un système qui autorise des raffinements architecturaux successifs jusqu'à obtenir un système temps réel qui optimise la surface de silicium et la consommation. Les méthodes basées sur la cosimulation (par exemple l'outil Felix de Cadence) nécessitent de déterminer préalablement un partitionnement logiciel/matériel des fonctions de l'application puis de développer des modèles précis de simulation. Ces techniques de cosimulation permettent de valider un système avant son intégration mais n'apportent pas une aide réellement efficace à la recherche d'architectures de niveau système qui réalisent les contraintes. Il y a donc un important besoin en outils d'aide à l'exploration de solutions architecturales dans le but de déterminer une spécification système précise qui supporte l'exécution de l'application et optimise les coûts en surface et en consommation.

2. Modélisation des applications et architecture générique

Plusieurs travaux sur la conception conjointe logicielle/matérielle abordent ce problème. Ils s'appuient en général sur des méthodes de partitionnement. Chaque méthode utilise un modèle de description des applications adapté au type d'applications ciblées. Par exemple, dans l'approche POLIS [1] orientée vers les systèmes réactifs temps réel, le modèle est basé sur les machines d'états finis hiérarchisées. Dans l'approche présentée dans [2] le modèle utilisé est le langage SDL adapté à la description de protocoles de communication. Généralement, les applications de traitement du signal contiennent une partie importante d'opérations sur des flots d'échantillons. Aussi, les approches orientées vers ce type d'applications (par exemple [3]) utilisent une modélisation fonctionnelle par des graphes de flots de données. Le modèle utilisé dans CODEF pour décrire le comportement de l'application et ses contraintes temporelles est basé sur ce formalisme.

Le type d'architecture ciblée par les outils de prototypage système est un facteur important pour leur utilisation effective. Il doit être en adéquation avec le type d'applications visées et permettre de construire des spécifications robustes. Une spécification système est robuste si les étapes successives de raffinements, nécessaires jusqu'à obtenir une spécification détaillée avant implémentation, n'entraînent pas la modification de cette spécification système du fait d'une estimation imprécise des paramètres des différents éléments architecturaux. Il est donc primordial, pour réduire les effets de ces modifications sur le *time-to-market*, de considérer et de déterminer avec précision toutes les caractéristiques architecturales de la spécification système qui ont un impact direct sur les performances et les coûts de la réalisation finale. Comme dans l'approche Coware [4], le modèle d'architecture de CODEF est basé sur une architecture parallèle hétérogène mais au lieu de considérer des communications points à points, CODEF utilise une interconnexion basée sur un modèle de bus qui facilite la réutilisation et la flexibilité. CODEF permet de déterminer les paramètres systèmes importants pour obtenir des spécifications systèmes robustes. Un système produit par CODEF peut comporter plusieurs instances de processeurs de types différents associés à des coprocesseurs ou à des accélérateurs matériels. Le système est caractérisé de façon

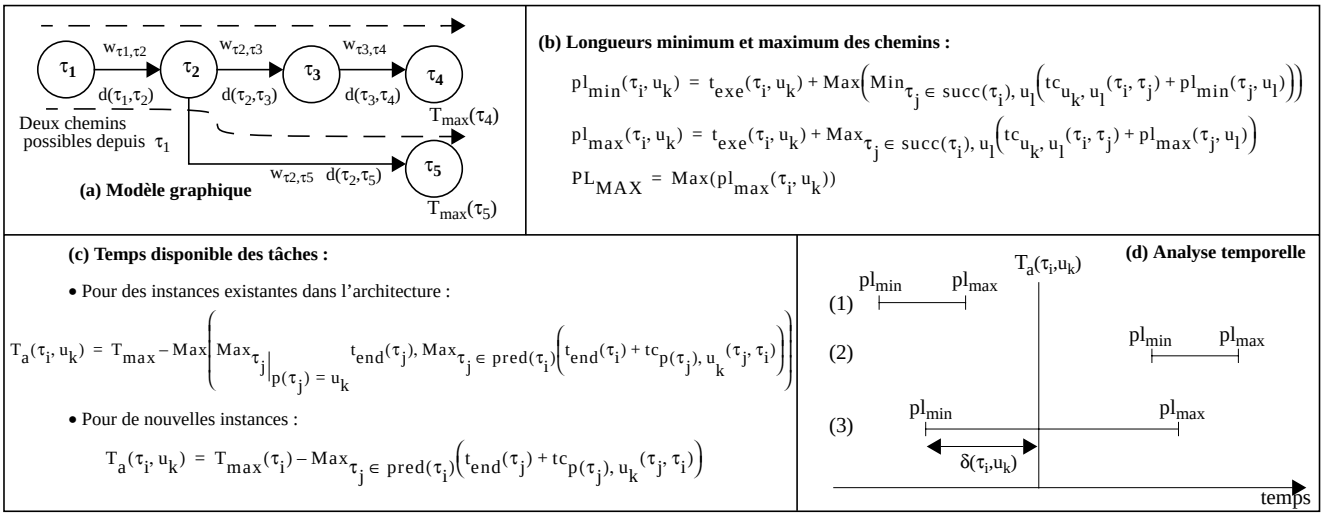

Fig. 1. Modèle graphique et analyse temporelle.

précise : temps d'accès et tailles des différentes mémoires (RAM, ROM, caches éventuels) des différents processeurs et des mémoires de communication, répartition en mémoire interne ou externe des données et des instructions de chaque processeur, mode de transfert par DMA ou contrôlé par CPU des données entre les différentes unités, interconnexion des unités (largeur des bus, ports d'entrée/sortie des unités reliées).

3. Flot de Conception dans CODEF

CODEF permet une exploration récursive d'un espace de conception système délimité par les contraintes de l'application (contraintes de temps et de surface) et les différents supports de réalisation possibles des tâches de la spécification. Cette exploration permet de construire un ensemble de solutions qui respectent les contraintes. Le concepteur peut ensuite sélectionner une solution particulière pour des optimisations supplémentaires. Ces optimisations sont réalisées de manière interactive par le concepteur. Pour ce faire, CODEF prend en entrée la description architecturale d'un système. Cette fonctionnalité permet d'une part, d'analyser rapidement des modifications proposées par le concepteur dans le but d'optimiser certaines caractéristiques (par exemple analyser l'effet d'une réduction de la taille des caches). D'autre part, il est aussi possible de réutiliser un système existant comme support de réalisation d'une nouvelle application, avec ou sans modifications architecturales.

4. Méthode de partitionnement

CODEF réalise un partitionnement/ordonnancement des différentes tâches τ_i qui composent la description flots de données (DFG) de l'application (Figure 1.a). Sur les arcs sont annotés la largeur en octets $w(\tau_i, \tau_j)$ et le volume en octets $d(\tau_i, \tau_j)$ des données échangées entre ces tâches. Sur les tâches terminales sont spécifiées des contraintes temporelles maximum de fin d'exécution de ces tâches. Les valeurs de ces contraintes sont propagées vers les tâches prédécesseurs. Lorsqu'une tâche a plusieurs successeurs, la contrainte la plus sévère est sélectionnée. L'algorithme utilisé opère en deux étapes [5]. Dans la première étape sont calculées des bornes minimum et maximum des longueurs temporelles des chemins issus de chaque tâche vers les tâches terminales (Figure 1.b). Ces calculs, réalisés itérativement à partir des tâches terminales, prennent en compte les temps d'exécution $t_{exe}(\tau_i,u_k)$ des différentes réalisations u_k (logicielles ou matérielles) possibles de chaque tâche et les éventuelles communications inter-tâches $tc_{u_k,u_l}(\tau_i, \tau_j)$. La deuxième étape réalise le partitionnement proprement dit. L'algorithme effectue un ordonnancement et une allocation pour les tâches ordonnançables.

4.1. Estimation des temps d'exécution

Les temps d'exécution des tâches sont dépendants de la structure mémoire des unités (présence d'un cache, mémoire interne et/ou externe) ainsi que du placement des tâches en mémoire. La durée d'exécution en cycles d'une tâche placée en interne r_{int} est fournie dans la librairie des implémentations des tâches. Pour chaque implémentation de tâche le taux d'accès en mémoire ρ_{ma} par rapport à r_{int} est fourni. Lorsque l'unité possède un cache, le calcul du temps d'exécution utilise le taux de défaut de cache c_{miss} et la taille des blocs de données c_{block} transmis entre la mémoire externe et le cache. La table 1 résume les cas considérés dans notre approche, où e_p représente le temps d'accès en mémoire externe et T_c la période d'un cycle d'horloge de l'unité.

Table 1 : Influence de la mémoire sur les temps d'exécution

Code	Data	Nombre de cycles d'exécution
Int	Ext	$r_{ext} = r_{int}\left(1 + \rho_{ma}\left(\left\lceil \frac{e_p}{T_c} \right\rceil - 1\right)\right)$
Ext	Ext	$r_{ext} = r_{int}\rho_{ma}\left\lceil \frac{e_p}{T_c} \right\rceil$
Int	Int	r_{int}
Cache	Cache	$r_{cache} = r_{int}\left(1 + \rho_{ma}c_{miss}c_{block}\left\lceil \frac{e_p}{T_c} \right\rceil\right)$

4.2. Discriminants temps et surface

Pour guider le partitionnement, un temps disponible

$T_a(\tau_i,u_k)$ pour chaque tâche (Fig. 1.c) et chaque unité u_k capable d'exécuter la tâche est évalué. Ce temps dépend de la présence d'une instance d'unité dans l'architecture en cours de construction. La valeur de $T_a(\tau_i,u_k)$ est calculée à partir des fins d'exécution $t_{end}(\tau_j)$ des prédécesseurs de τ_j, de la disponibilité de u_k ($p(\tau_j)$ correspond à l'unité qui réalise τ_j dans l'architecture) et des temps de communication. Les temps de communication sont évalués à partir de $w(\tau_i, \tau_j)$, de $d(\tau_i, \tau_j)$ et des débits les plus rapides des ports d'entrée/sortie des unités $p(\tau_j)$ et u_k. Puis, en fonction des longueurs des chemins et des contraintes de temps (Fig. 1.d), l'algorithme détermine des degrés d'urgence :

$$\gamma(\tau_i, u_k) = 1 - \frac{1}{\delta(\tau_i, u_k)} \times \frac{pl_{max}(\tau_i, u_k)}{PL_{MAX}}$$

Le rapport $pl_{max}(\tau_i,u_k)/PL_{MAX}$ permet de réduire l'influence du degré d'urgence pour les tâches proches des tâches terminales. La modélisation de la surface de silicium $S(\tau_i,u_k)$ [5] pour les tâches ordonnançables tient compte :

- de la surface du cœur de l'unité : surface d'instanciation pour une nouvelle unité dans l'architecture,
- de la surface fonctionnelle relative aux tailles de RAM et ROM pour mémoriser les données et le code d'une tâche si celle ci n'a pas été préalablement placée sur cette unité.

De plus, la surface est pondérée par la possibilité de réutiliser cette unité pour exécuter les tâches restantes. La surface est ensuite normalisée relativement aux tâches ordonnançables τ_j :

$$\theta(\tau_i, u_k) = 1 - \frac{S(\tau_i, u_k)}{Max_{u_m, \tau_j}(S(\tau_j, u_m))}$$

On définit alors le facteur discriminant mixte :

$$\Omega(\tau_i, u_k) = (1-\alpha)\gamma(\tau_i, u_k) + \alpha \times \theta(\tau_i, u_k)$$

où α est un paramètre défini par l'utilisateur.

4.3. Heuristique de partitionnement

A partir d'une architecture A_n (qui peut être vide), l'heuristique de partitionnement construit à l'étape n des listes avec les différents couples $<\tau_i,u_k>$ correspondant aux tâches ordonnançables (Fig. 2). Dans le cas d'unités logicielles, la répartition dans les listes tient compte de la présence d'un éventuel cache dans la hiérarchie mémoire ou de la localisation de la tâche dans la mémoire interne ("on-core") ou externe ("off-core") de l'unité. Une localisation en mémoire interne ne peut être envisagée que si l'espace libre dans cette mémoire est suffisant pour contenir tout le code et les données de la tâche. Lors d'une affectation d'une tâche sur une unité, l'algorithme met à jour cet espace disponible pour la suite du partitionnement. Cette structure mémoire est prise en compte pour ajuster le temps d'exécution de la tâche conformément au modèle du paragraphe 4.1, avant le placement dans les listes.

Une fois ces listes remplies, l'algorithme sélectionne la tâche la plus critique ($Max_{u_k}(T_a(\tau_i, u_k) - pl_{min}(\tau_i, u_k))$ est minimum) et recherche dans ces listes la "meilleure" implémentation de cette tâche en privilégiant la réutilisation (Fig. 2.). Les deux paramètres T_{reuse} et T_{new}, dont les valeurs sont comprises entre 0 et 1, permettent d'orienter les choix effectués par l'algorithme. Le premier correspond à un seuil de réutilisation des instances contenues dans l'architecture en cours de construction. Plus il est élevé, plus les réalisations possibles des

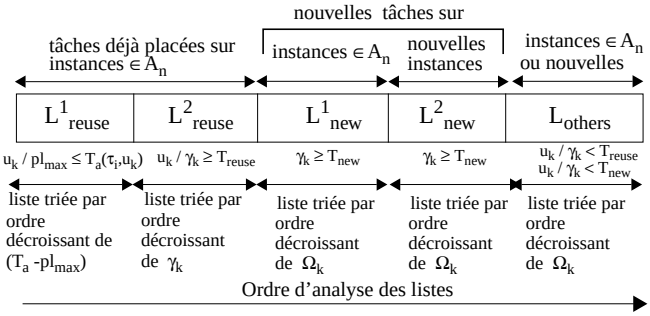

Fig. 2. Construction des listes avec les couples $<\tau_i,u_k>$.

tâches sur ces instances doivent être rapides.

Lorsqu'une tâche n'est exécutée par aucune instance de l'architecture A_n, du fait des contraintes temporelles ou de fonctionnalités insuffisantes, l'algorithme ajoute une nouvelle instance à l'architecture ou complète une instance existante (coprocesseur par exemple). La sélection de cet enrichissement est guidée par la valeur du deuxième paramètre T_{new} : une valeur élevée conduit à sélectionner des réalisations rapides des tâches. Des valeurs faibles de T_{new} et T_{reuse} offrent a priori plus de possibilités de choix et l'algorithme effectue alors une recherche de compromis temps/surface. Dans le cas d'une utilisation interactive de CODEF, les valeurs de ces paramètres sont fixées par l'utilisateur. Pour effectuer une exploration récursive de l'espace de conception, l'algorithme de partitionnement calcule les variations minimum des valeurs de ces deux paramètres qui conduisent à déplacer dans une liste différente la "meilleure" réalisation de la tâche la plus critique. Ensuite par récursivité, on obtient plusieurs solutions architecturales.

5. Optimisation interactive

CODEF utilise en entrée la description d'un système optionnel prédéfini, résultant par exemple de l'exploration de l'espace de conception par l'approche récursive ci-dessus. Un système prédéfini contient :

- un ensemble d'instances logicielles ou matérielles avec leur caractéristiques (période d'horloge, taille des mémoires RAM, ROM et leur localisation interne ou externe, présence d'un cache par exemple),
- une liste optionnelle d'assignations de tâches sur les instances pour privilégier leur exécution par ces unités,
- une description optionnelle d'une interconnexion des instances par des bus et des mémoires double port ou FIFO.

Comme CODEF essaye de réutiliser au mieux les ressources de l'architecture en cours de construction (listes L^1_{reuse}, L^2_{reuse} et L^1_{new}), les instances et les assignations ainsi déclarées dans le système prédéfini permettent au concepteur d'orienter les choix réalisés par l'algorithme dans le but de proposer des optimisations.

6. Résultats

Pour illustrer ces principes et les différentes fonctionnalités de CODEF, la conception d'un décodeur audio AC3 [6] est étudiée. Le graphe simplifié de ce décodeur est donné sur la Fig. 3. La contrainte temporelle sur toutes les sorties est fixée

à 3 ms. Les caractéristiques des réalisations des différentes tâches sont détaillées dans la table 2. On considère un seul type de processeur Sw capable d'exécuter l'ensemble des tâches. La tâche *BitAlloc* possède une réalisation entièrement logicielle sur Sw, une réalisation mixte sur Sw avec un coprocesseur et une réalisation entièrement matérielle. Toutes les

Table 2 : Temps d'exécution (cycles), tailles Rom/Ram (octets) et surface (mm^2)

	EXP	BA	DM	DC	RM	ITDAC
Rom/Ram	774/258	1092/200 copro: 800/200	274/150	120/48	20/0	776/384
exécution SW[a]	8280	55200 copro[b]: 23200	28440	7350	3120	15600
unités HW	HW_exp	HW_ba	HW_dm	HW_dc	HW_rm	HW_itdac
exécution HW	1045	5120	3370	1400	1020	3160
surface HW	0.023 mm^2	0.35 mm^2	0.13 mm^2	0.09 mm^2	0.04 mm^2	0.86 mm^2

a.La surface du coeur du SW est 3.85 mm^2
b.La surface du coprocesseur est 0.11 mm^2

tâches ont des réalisations matérielles sur des accélérateurs différents. L'algorithme récursif exécuté avec un système initial vide, fournit 50 solutions, générées en moins de 7.5 secondes sur une station Sun Ultra 5, dont 13 sont illustrées sur la Fig.4 par des cercles. La solution (7) semble optimum car

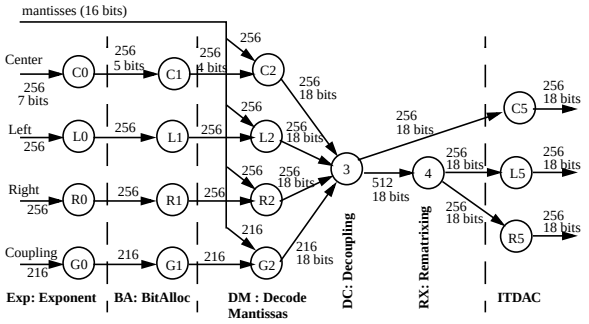

Fig. 3. Graphe simplifié du décodeur audio AC3

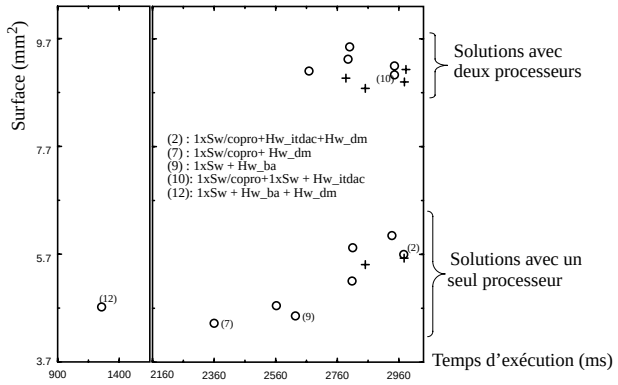

Fig. 4. Résultats de l'exploration spatiale

nous n'avons trouvé aucune solution à surface plus faible satisfaisant la contrainte de 3 ms. Les assignations des tâches aux unités de la solution (2) sont : Exp, BA, RM sur SW avec coprocesseur, itdac, BA, DM sur des accélérateurs. La représentation graphique de l'ordonnancement des tâches sur cette architecture est donnée sur la Fig. 5. Les solutions notées avec un '+' de la Fig. 4 sont générées avec le système (2) comme système initial. Ces solutions sous-optimales illustrent que l'algorithme n'est pas capable de rectifier des choix inappropriés introduits initialement. Cependant, si on enlève manuellement du système initial les unités HW_itdac, HW_ba et les assignations des tâches correspondantes, l'algorithme trouve directement la solution (7). Cette possibilité illustre que CODEF fournit une aide efficace pour l'optimisation d'architectures systèmes basée sur l'expérience du concepteur.

Fig. 5. Diagramme de Gantt d'AC3 sur le système (7)

7. Conclusion

L'outil CODEF permet d'aider le concepteur dans la détermination des spécifications d'un système temps réel à partir d'une description flots de données d'une application. Dans les étapes de raffinements de cette spécification qui conduisent à la réalisation effective, les caractéristiques des différents éléments architecturaux du système deviennent plus précises et peuvent s'écarter des valeurs estimées initiales. Avec la possibilité qu'offre CODEF de prendre en compte l'architecture d'un système prédéfini, le concepteur peut s'assurer à tout moment de la validité du système en cours de conception vis à vis des contraintes ou peut effectuer les ajustements minimum nécessaires. L'objectif de CODEF est de déterminer des architectures de systèmes qui satisfont les contraintes temporelles et optimisent la surface. Les extensions en cours d'étude de la méthode concernent principalement, l'optimisation des communications et la prise en compte de la partie contrôle de la spécification d'une application.

Références

[1] T. Cuatto, C. Passerone, L. Lavagno, A. Jurecska, A. Damiano, C. Sansoe, A. Sangiovanni-Vincentelli. *A Case Study in Embedded System Design: an Engine Control Unit*, In Proc. of Design Automation Conference, 1998.

[2] J.M. Daveau, G.F. Marchioro, T. Ben_Ismail, A.A. Jerraya. *COSMOS: An SDL Based Hardware/Software Codesign Environment*, dans Hardware/Software Co-Design and Co-Verification, Kluwer Academic Publishers, édité par J.M. Berge, O. Levia, J. Rouillard, 1997

[3] A. Kalavade, E. Lee. *The extended partitioning problem: hardware/software mapping and implementation-bin selection*, Proceedings Int. Workshop on Rapid System Prototyping, Chapel Hill, NC, June 7-9, 1995.

[4] K Van Rompaey, D. Verkest, I. Bolsens, H. De Man, *CoWare - A design environment for heterogeneous hardware/software systems*, Proc. EURO-DAC'96 with EURO-VHDL'96, 1996.

[5] L. Bianco, M. Auguin, A. Pegatoquet, *A prototyping method of embedded real time systems for signal processing applications*, Euromicro'99, Milan, 8-10 sept. 1999.

[6] *Digital Audio Compression Standard (AC3)*, Advanced Television Systems Committee, 20 Dec. 1995.