Estimations et métriques au niveau système pour la conception conjointe logiciel/matériel

Hervé THOMAS, Jean-Philippe DIGUET, Jean-Luc PHILIPPE

LESTER, Université de Bretagne Sud
rue de Saint MAUDE, 56100 Lorient, France

Herve.Thomas@univ-ubs.fr, Jean-Philippe.Diguet@univ-ubs.fr, Jean-Luc.Philippe@univ-ubs.fr

Résumé – Cet article s'inscrit dans le thème de l'adéquation application-architecture. La méthodologie proposée couvre les étapes hautes du flot de conception conjointe, commençant au niveau de la spécification et s'arrêtant avant l'étape de partitionnement. Le but est de fournir au concepteur et à l'étape de partitionnement des informations utiles afin de construire une architecture "ad hoc" optimisée. Ainsi les estimations sont établies sans à priori sur l'architecture cible. Les potentiels d'optimisation existants entre les fonctions sont pris en compte pour obtenir un coût global et dynamique de l'application.

Abstract – These paper comes within the framework in the application-architecture matching. The proposed methodology covers the upper part of the co-design flow which is located before the partitioning step. The issue is to provide the designer and to the partitioning step with useful information in order to design an ad hoc architecture. Also, the estimations are computed without knowledge of the implementation. The optimisation potential existing between the function are taking into account to obtain a global and dynamic cost of the application

1. Introduction

La réalisation des applications du traitement du signal tend vers des architectures hétérogènes pour des systèmes complexes composés d'un nombre important de fonctions de différentes natures. Le concepteur est alors confronté à un espace de solutions très vaste tant au niveau architectural que algorithmique. Outre le choix classique de partitionnement logiciel /matériel, le *Codesign*[1](cf Fig. 1) doit décider de la nature de l'architecture cible (DSPs, FPGAs, ASICs, ASIPs, RISCs, Hiérarchie de mémoires, etc.). Compte tenu de l'impact des choix au niveau système (spécification et algorithme) sur le coût final et les performances atteintes, il est important d'une part d'estimer au niveau système de façon rapide le coût (temps, surface et consommation).

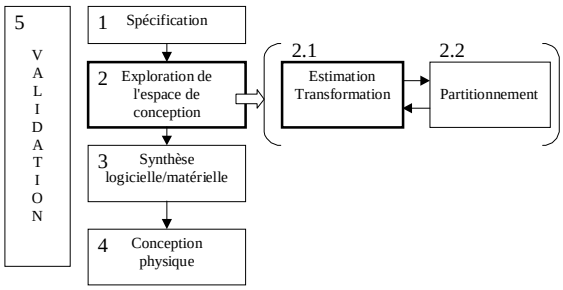

Fig. 1: Flot de conception conjointe (*Co-design*)

D'autre part, il est nécessaire d'aider par des métriques la conception de l'architecture cible. Nous présentons, dans cet article, la partie "haute" du flot de conception conjointe logiciel/matériel située avant le partitionnement (étape 1 et 2.1 fig. 1). L'objectif est de fournir un coût relatif et un ordre de placement des différentes fonctions sur l'architecture à construire. A notre connaissance, peu de méthodes d'estimations se situent avant la boucle de partitionnement. Cependant, des estimations existent dans le cas particulier de la mémoire[2][10] et des études de calculs de métriques de rapprochement au niveau système (closeness metrics[3]) ou à des niveaux inférieurs [4][5][6]. Par contre, il existe de nombreuses méthodes d'estimations durant la phase de partitionnement avec une architecture cible donnée [7][8]. L'originalité de notre approche repose sur cinq points principaux : (i) Une analyse macroscopique de la spécification afin d'obtenir deux types d'informations nécessaires sur l'orientation des différentes fonctions de l'application qui peut–être orientée traitement, contrôle ou mémoire (TCM) et leur coût en terme de valeurs opératives. (ii) Le second point est le niveau d'abstraction qui fournit des estimations décorrelées de tout modèle architectural spécifique. (iii) Nous calculons et utilisons des courbes de coût dynamique au lieu d'un coût fixe basé sur une bibliothèque de composants. (iv) Le point suivant concerne l'estimation du coût qui est interdépendant. En effet, le coût dynamique d'une fonction est calculé en tenant compte de ces interactions avec les autres fonctions afin de tirer le meilleur profit du potentiel de réutilisation et d'optimisation. (v) Pour la construction de l'architecture, après l'étape d'estimation et durant le partitionnement, il est crucial de placer les différentes fonctions dans un ordre qui privilégie un choix architectural basé sur l'adéquation avec les fonctions les plus critiques. Ainsi, notre méthode fournit à l'étape suivante (partitionnement), l'ordre dans le quel les

différents fonctions de l'application doivent être prises en compte pour construire graduellement l'architecture "ad hoc". Dans ce document, nous décrivons chaque étape du flot de conception ainsi que des exemples issuesde l' application de traitement du signal G.729 (codage de la parole).

2. Flot de conception

Le flot de conception (Figure 2) représente le méthodologie à suivre pour obtenir les informations nécessaires au concepteur pour effectuer ses choix algorithmiques, architecturaux ou méthodologique (orientée contrôle, mémoire ou traitement) au niveau du choix des outils de synthèses d'architectures adaptés.

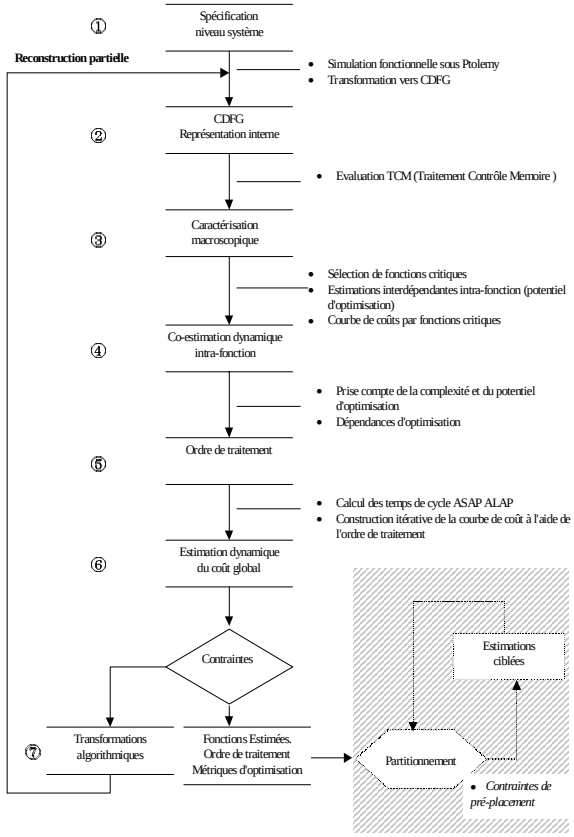

Fig. 2: Flot de conception

2.1 Spécification niveau système

La spécification au niveau système est une description de l'application avec une granularité définie par l'utilisateur. Dans un premier temps, elle correspond à une découpe naturelle en fonctions (Auto-corrélation, Fenêtrage, etc.). Dans un second temps, une décomposition automatique est réalisée pour obtenir une granularité adaptée (compromis nombre de nœuds/temps d'estimation). Avant d'effectuer les caractérisations et autre estimations, nous procédons à une simulation fonctionnelle de la spécification du système afin d'éviter les erreurs de conception dues à une mauvaise spécification. Pour réaliser cette étape, nous avons choisi comme environnement de spécification le logiciel Ptolemy [9] sous lequel nous avons défini un modèle de spécification générique et hiérarchique.

2.2 CDFG

Dans le but de faciliter la manipulation de la spécification du système décrit sous PTOLEMY, nous transformons cette dernière en une représentation interne sous forme de graphe de flots de données et de contrôle (CDFG) G={V, E}. Chaque nœud du graphe $v_i \in V$ correspond soit à une opération de traitement (v_i^t), soit une opération de contrôle (v_i^C), un accès mémoire (v_i^M), ou à un ensemble de nœud (v_i) dans le cas d'un graphe hiérarchique. Un arc $e_{i,j} \in V$ donne la direction de la dépendance d'exécution du nœud v_i au nœud v_j. La granularité du CDFG est limitée afin de respecter le compromis entre le temps de calcul et la précision des estimations. Cela signifie que deux nœuds au plus bas niveau de granularité ne peuvent être décomposés afin de trouver une optimisation entre les graphes qui composent chaque nœud.

2.3 Caractérisation macroscopique

L'objectif de la caractérisation macroscopique est de définir la nature du système spécifié. Dans un premier temps, nous évaluons, au niveau des fonctions de base, le nombre de traitements, de contrôles et de mémoires (TCM) séparément pour chaque fonction. La partie mémoire comprend trois paramètres: i) les accès mémoires (AM) ii) le nombre de points mémoires (#M) nécessaires pour l'exécution d'une fonction iii) les traitements mémoires (TM) pour le calcul des indices de tableaux (ex: tab1[n-k]). Pour la caractérisation de la mémoire, nous nous inspirons des travaux réalisés à l'IMEC [11]. Ainsi, les valeurs obtenues reflètent le coût TCM de chaque fonction (caractérisation intra-fonction).

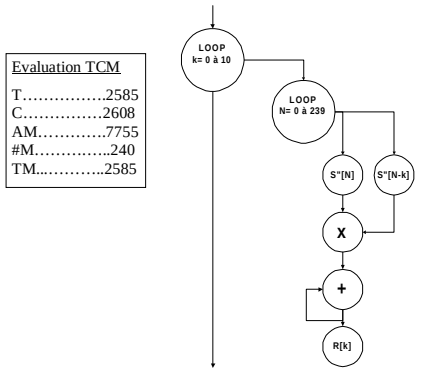

Fig. 3: Caractérisation Macroscopique de la fonction auto-corrélation

Dans un second temps, le résultat de la caractérisation nous permet de sélectionner un groupe de fonctions dites "critiques" (Fs_i) parmi l'ensemble des fonctions du système. Le nombre de fonctions critiques sélectionnées est basé sur un compromis complexité/précision. Si le temps de calcul (i.e. le nombre de nœud maximum dans le CDFG) autorisé est suffisant, toutes les fonctions du système sont incluses dans le groupe des fonctions critiques. Cela signifie que nous considérons seulement les fonctions qui conduiront à la définition de l'architecture, c'est à dire les fonctions les plus coûteuses en termes de métriques TCM. Nous supposons que les fonctions non sélectionnées pour notre flot de conception

ont un faible impact sur la réalisation du système. Ainsi, le concepteur utilisera l'architecture définie par les fonctions les plus critiques.

2.4 Co-estimation dynamique intra-fonction

Etant donné un groupe de fonctions critiques, l'objectif de la co-estimation dynamique est de fournir une courbe de coût du nombre d'opérations simultanées (TCM) en fonction de contrainte de temps. Notons que cette contrainte est exprimée en nombre de cycle d'horloge (CCC *Constraint Clock Cycle*). Actuellement, nous nous intéressons à la co-estimation dynamique des traitements et des accès mémoires, nous considérerons plus tard le cas de la partie contrôle.

Notre approche se décompose en deux étapes. Premièrement, les différentes branches (sous fonctions) de la fonction sont identifiées et classées par ordre de criticité (somme du nombre d'opérations et des accès mémoires sur le chemin critique). Deuxièmement, cet ordre est utilisé pour construire de manière itérative la courbe de coût de la fonction. Le coût de chaque branche est alors estimé par ordre décroissant de criticité. Ces estimations tiennent compte de la distribution des opérations utilisées pour les branches précédemment estimées (plus critiques). Le but étant de minimiser le parallélisme à l'intérieur d'une fonction. Ainsi, nous favorisons la réutilisation de blocs de traitement des données (et /ou accès mémoires) qui sont présents plusieurs fois dans le graphe d'une fonction.

Les opérations du graphe peuvent être soit des opérations élémentaires soit des opérations complexes caractérisées par leur coût. Cela dépend du compromis temps de calcul/ précision des estimations. Pour des opérations élémentaires, nous distinguons deux familles d'opérations. La première concerne les opérations de traitement, où deux sous-types d'opérations ont été définis: UAL (+, -, Comp, Logique) et MAC(*, *+, *-). La seconde famille concerne les accès mémoires qui sont soit locaux soit globaux. Nous définissons comme local tous les accès à des données qui ont déjà été lues ou écrites au moins une fois dans la fonction. Inversement, les accès mémoires globaux correspondent à un premier accès d'une donnée. Au niveau système sans définition d'architecture cible, nous n'avons aucune connaissance sur la fréquence d'horloge. Ainsi, nous supposons que chaque type d'opérations élémentaires nécessite un temps cycle. Nous pensons que les estimations basées sur de telles hypothèses sont suffisamment informatives pour guider correctement le concepteur vers une architecture optimisée.

Comme nous l'avons décrit précédemment, la co-estimation dynamique est donnée en deux étapes. Ci-dessous, nous expliquons le principe d'estimation du coût d'une branche d'une fonction pour différents CCC (Fig. 4). Cet algorithme est répété tant que le nombre des ressources de chaque type d'opérations n'est pas minimal (égale à 1). C'est à dire que le coût de la fonction est calculé jusqu'à obtenir une solution totalement séquentielle. Si la fonction est constituée de plusieurs branches, alors l'algorithme doit tenir compte de la distribution des opérations utilisée pour les branches précédemment estimées.

1. *Calcul des dates ASAP et ALAP de chaque nœud pour une CCC donnée.*
2. *Calcul du nombre probable d'opérations simultanées pour chaque temps de cycle Nproba(cycle).*
3. *Calcul du nombre moyen d'opérations simultanées.*
4. *Analyse des "Bottlenecks". Grâce à la courbe Nproba(cycle$_i$), les "bottlenecks" sont détectés et analysés afin de vérifier si le nombre moyen d'opérations simultanées est suffisant pour exécuter l'itération complète pour une CCC donnée. Si le nombre moyen est trop petit, il est alors incrémenté. Notons que l'affection "un cycle par opération" rend plus facile cette tâche.*
5. *Pseudo-list scheduling à haut niveau où le temps est donné en nombre de cycles. Cette étape permet d'assigner un cycle par opération. Ainsi, nous connaissons à quel cycle une ressource donnée peut être réutilisée.*

Fig. 4: Algorithme d'estimation de coût

Pour illustrer notre approche, nous prenons une fonction de l'application du codage de la parole G.729 (Fig. 5) qui possède trois sous-fonctions (branches) SF1, SF2, SF3. Ainsi, la première étape consiste à évaluer la criticité de chacune d'entre elles. Les sous-fonctions seront estimées dans cet ordre. Cela signifie que la première sous-fonction imposera ces contraintes à l'estimation des sous-fonctions suivantes. Dans notre cas, SF2 est la plus critique. Par conséquent, elle est estimée en première position sans aucune contrainte. Après quoi, SF1 et SF3 peuvent être estimées en tenant compte de l'optimisation liée à la distribution des ressources estimées pour SF2. La même technique sera utilisée durant la recherche du potentiel d'optimisation entre toutes les fonctions de l'étape d'estimation dynamique du coût inter-fonction. Ainsi, nous pouvons calculer le profil cycle par cycle du coût global, et ainsi caractériser le coût de l'application.

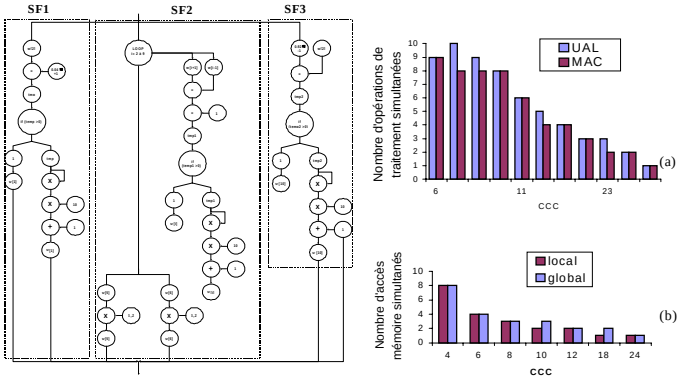

Figure 5: Fonction G.729 (a) courbe de coûts des opérations de traitement (b) courbes de coûts des accès mémoires

2.5 Ordre de traitement

Le problème à ce niveau du flot de conception est de trouver l'ordre dans lequel les fonctions critiques doivent être analysées. Dans notre approche, nous effectuons une première classification en fonction de la complexité (coût TCM). Puis, dans un second temps, nous affinons cet ordre

de traitement à l'aide de métriques la nature des liens et potentiels d'optimisations (contraintes de placement) existants entre les fonctions. Ces métriques sont appelés métriques de dépendances inter-fonction. Ils sont similaires à ceux qui sont proposés dans [3]. Pour le potentiel d'optimisation de la taille et des accès mémoires, nous utilisons les techniques décrites dans [11]. Cette étape inclue aussi une optimisation sur l'ordre d'exécution (Fig. 6). Le but étant de guider le concepteur et les outils: si des opportunités de réutilisation de ressources sont mises en évidence, des dépendances virtuelles sont ajoutées sur le CDFG. Bien sûr, de telles contraintes sont ajoutées avec le respect des CCC et le potentiel d'optimisation avec les autres fonctions. Comme nous pouvons le voir sur la figure 5, la fonction Fs_1 et Fs_3 accèdent en lecture au même vecteur A, tandis que Fs_2 lit le vecteur B. Afin d'optimiser les accès mémoire, nous introduisons des dépendances virtuelles d'optimisation sur le CDFG qui spécifie que Fs_3 doit être exécutée avant Fs_2. Ce qui conduit à optimiser l'accès mémoire du vecteur A.

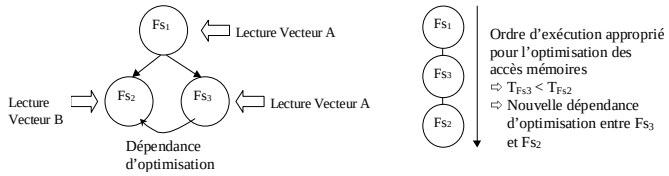

Fig. 6: Optimisation de l'ordre d'exécution

2.6 Co-estimation dynamique inter-fonction

Etant donnée l'ordre de traitement des fonctions, la co-estimation des fonctions critiques est maintenant possible. Comme définie précédemment pour la co-estimation intra-fonction, les courbes de coût seront calculées pour chaque fonction en tenant compte du potentiel d'optimisation des données et des ressources. Le coût pour chaque fonction est donné pour différents CCC.

2.7 Estimation dynamique du coût global

Avec l'ordre de traitement, les dépendances virtuelles ajoutées pour l'optimisation des ressources, la réutilisation des données et les courbes de coût de chaque fonction Fs_i, nous avons suffisamment d'informations pour établir le profile du coût globale. Cela signifie le coût par cycle de l'ensemble des fonctions critiques du premier au dernier cycle pour une contrainte de débit donnée. L'estimation du coût global est construite de manière itérative. Le problème est, pour une CCC définie par le concepteur, comment répartir les cycles de la contrainte de temps sur l'ensemble des fonctions critiques. Ainsi en fonction de la distribution du coût, il peut être intéressant d'augmenter le parallélisme inter-fonction et diminuer le parallélisme intra-fonction ou vice versa. Dans un premier temps, le principe de la distribution des cycles d'horloges pour chaque fonction est proportionnelle à la criticité des fonctions. Puis le profile du coût globale est analysé afin de détecter les « *bottlenecks* ». Ensuite, les temps cycles accordés aux fonctions critiques sont redistribués dans le but de rendre plate la courbe de coût globale. A l'heure actuelle, le calcul du coût est basé uniquement sur l'aspect mémoire et traitement qui peuvent être séquentiel ou parallèle. La décision de favoriser la mémoire ou la partie traitement est donné à l'aide des métriques TCM.

2.8 Transformations algorithmiques

La plupart des primitives en traitement du signal possèdent plusieurs algorithmes, par conséquent, nous nous basons sur ces possibilités pour proposer au concepteur des nouvelles solutions afin augmenter les performances du système. Cette étape n'étant pas un passage obligatoire, elle n'est utilisée que dans les cas où les contraintes ne sont pas satisfaites ou bien si le concepteur souhaite tester une autre combinaison algorithmique.

3. Conclusion

Dans cet article, nous proposons un flot d'estimation au niveau système sans à priori sur l'architecture cible. Nous avons présenté des techniques d'estimation permettant i) dans un premier temps, de sélectionner un groupe de fonctions critiques (caractérisation macroscopique) ii) puis, de caractériser le coût d'une fonction sous forme de courbe de coût dynamique en tenant potentiel d'optimisation existant entre chacune des branches de la fonction. Par extension de cette technique et connaissant la criticité de chaque fonction, nous pouvons estimer le coût dynamique inter-fonction. Ainsi, nous calculons le profil global du coût de l'application. Ce qui se traduit pour le concepteur par coût relatif du système suffisamment précis pour choisir la meilleure architecture.

Références

[1] "Hardware/software Co-design study report", MCC/OMI, 1997

[2] S.Wuytack, J.Ph.Diguet, F.Catthoor, H.De man, "Formalized methodology for data reuse exploration for low-power hierarchical memory mappings", IEEE Trans. on VLSI Systems, Vol. 6, 1998

[3] F.Vahid and D.D.Gajski "Closeness metrics for system-level functional partitioning", EDAC, september 1995, pp 328-333

[4] R. Camposano and R. Brayton, "Partitioning before logic synthesis" Proc. ICCAD, 1987

[5] E. Lagnese and D. Thomas, "Architectural partitioning for system level synthesis of integrated circuits" IEEE Trans. CAD, 1991.

[6] X. Xiong, E. Barros, and W. Rosentiel, `"A method for partitioning UNITY language in hardware and software", EuroDAC, 1994.

[7] J. Madsen, and al. ,"LYCOS: the Lyngby Co-Synthesis System", Design Automation for Embedded Systems,1997

[8] J. Henkel, R. Ernst. "A Hardware/Software Partitioner Using a Dynamically Determined Granularity." 34th DAC, 1997.

[9] Almagest- Volume 1, "Ptolemy 0.7 Users Manual", http://ptolemy.eecs.berkeley.edu

[10] Balasa, F.; Catthoor, F., and De Man, H. "Background memory area estimation for multidimensional signal processing systems.", IEEE Trans. VLSI Systems. Vol. 3: (2)157-172; 1995.

[11] E.De Greef, F. Catthor and H. De Man, " Array placement for storage size reduction in embedded multimedia systems".