VESTIM : Une méthode d'estimation de performances pour une implémentation optimisée d'applications sur processeurs de traitement du signal

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

[1]Laboratoire 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 Inc.
505, route des Lucioles, 06560 Valbonne, France

alain.pegatoquet@vlsi.com, auguin@i3s.unice.fr, kwiatkow@i3s.unice.fr
emmanuel.gresset@vlsi.com, bianco@i3s.unice.fr

Résumé – Les compilateurs C pour processeurs DSP actuellement disponibles sont généralement incapables de générer un code assembleur respectant les contraintes temps réel fortes des systèmes embarqués. D'autre part, programmer un DSP directement en assembleur est une situation de plus en plus inacceptable. Notre approche se propose de fournir des estimations logicielles qui aident le programmeur au développement rapide d'applications sur DSP. Le programmeur dispose d'une évaluation des performances du code généré par le compilateur ainsi que d'une estimation d'un code assembleur optimisé. Nous comparons ces estimations avec des mesures de performances dans le pire cas obtenues en utilisant une approche statique.

Abstract – Current DSP C compilers are generally unable to generate efficient assembly code that respects tight real-time constraints. On the other hand, programming DSP applications by hand in assembly language is becoming unacceptable as applications increase in complexity. We present a software estimation methodology from a C description that helps programmers for a rapid development of DSP applications. Our tool VESTIM provides both a performance evaluation for assembly code generated by the compiler and an estimation of an optimized assembly code. Results are compared with measures of performance in the worst case obtained using a static approach based on ILP.

1. Introduction

Les processeurs de traitement du signal (DSP) sont particulièrement utilisés dans le domaine des systèmes embarqués car ils offrent un bon compromis en taille de silicium, consommation et vitesse. Ces processeurs disposent d'une architecture et d'un jeu d'instructions optimisés pour une exploitation temps réel efficace des traitements nombreux et répétitifs inhérents aux applications du traitement numérique du signal. Les progrès technologiques récents ainsi que l'utilisation d'architectures parallèles permettent aux constructeurs de proposer des DSPs dont la puissance de calcul peut atteindre plusieurs centaines de MIPS (350 MIPS pour le processeur PALM [1]). Bien que la plupart des normes liées aux applications de télécommunications soit fournie avec un programme C par les organismes agréés tels que l'ETSI (European Telecommunications Standardization Institute) ou l'ITU (International Telecommunication Union), l'utilisation des compilateurs C pour DSP est encore peu répandue. En effet, ces derniers génèrent le plus souvent un code assembleur dont les performances en nombre de cycles et en taille de code ne respectent pas les contraintes fortes de temps réel et de surface des systèmes embarqués. Les causes de cette inefficacité sont multiples et sont détaillées en [5]. C'est pour cette raison que les applications sont en grande partie codées manuellement en assembleur. Or, le temps de codage manuel d'une application de traitement du signal est de plus en plus long et délicat compte tenu de l'augmentation de la complexité des applications mais aussi des architectures DSPs cibles récentes (VLIW, dual MAC[1] par exemple). Cette situation devient par conséquent inacceptable si l'on souhaite respecter les contraintes toujours plus fortes de "time-to-market".

2. Une approche de haut niveau

La réduction du temps de développement d'une application de traitement du signal passe par une utilisation plus systématique du compilateur C fourni avec le DSP. L'objectif de l'outil d'estimation de performances VESTIM est d'éviter le plus possible d'écrire du code assembleur en aidant le programmeur à mieux utiliser le compilateur C. VESTIM fournit deux types d'estimation : (i) une évaluation de performance du code assembleur généré par le compilateur, (ii) une estimation de la performance d'un code assembleur optimisé, i.e. comme s'il avait été écrit à la main. Ces résultats sont obtenus sans aucune simulation du code assembleur généré permettant un gain de temps considérable. Grâce à la première estimation, il est possible de déterminer très rapidement les parties de l'application les plus coûteuses en nombre de cycles. C'est sur ces parties que le travail d'optimisation doit être réalisé en priorité. La seconde estimation permet quant à elle d'avoir une idée de la qualité du code assembleur généré. En appliquant un ensemble de règles d'écriture du code C (certaines générales, d'autres plus

spécifiques à un processeur donné), nous avons montré dans [9] qu'il est possible d'améliorer de manière significative les performances du code généré. Nous comparons ces estimations avec des mesures de performances dans le pire cas obtenues en utilisant une approche statistique développée par [4].

3. Méthode d'estimation logicielle

3.1 Modèle de calcul du temps d'exécution

Le modèle couramment utilisé pour calculer le temps d'exécution d'une application est le suivant (où N est le nombre de bloc de base du programme) :

$$T_{exe} = \sum_{i=0}^{N} x_i c_i \quad (1)$$

Les x_i représentent le nombre d'exécutions de chaque blocs de base, alors que les c_i dénotent le temps d'exécution associé à chaque bloc de base sur le processeur cible. Un bloc de base est un segment de programme dont le seul point d'entrée est la première instruction et le seul point de sortie est la dernière instruction [7]. Notons que seuls les x_i sont indépendant de l'architecture cible. Le modèle défini en (1) suppose que l'exécution d'un bloc de base prend toujours le même nombre de cycle, ce qui est globalement le cas pour la classe de DSP que nous considérons. Cependant, ce temps peut varier sensiblement suivant les propriétés architecturales du processeur cible (pipeline, mémoire cache, etc.) [6]. Trouver le temps d'exécution dans le pire cas (respectivement dans le meilleur cas) revient à maximiser (respectivement à minimiser) cette expression. Malheureusement, les méthodes existantes pour déterminer le temps d'exécution dans le pire cas sont généralement très contraignantes pour l'utilisateur et souvent inadaptées aux besoins industriels [12]. L'une des principales contraintes, d'un point de vue industriel, est la rapidité d'obtention des estimations. En conséquence, pour le calcul des x_i notre choix s'est porté dans un premier temps sur une approche dynamique (statistique) basée sur l'exécution de séquences de test représentatives. La Figure 1 montre que l'outil d'estimation est composé de deux parties distinctes.

3.2 Une approche hybride

Le « *front-end* » a pour objectif de déterminer les informations dynamiques, c'est-à-dire le nombre d'exécution x_i de chaque instruction C. Ces informations dynamiques sont obtenues en exécutant le code C de l'application avec des séquences de test sur le processeur hôte (un PC ou une station). Les séquences de test permettent également de valider les optimisations successives apportées au code C.

Pour le calcul des c_i, une première approche consiste à effectuer une simulation des instructions assembleurs de chaque bloc de base en utilisant un simulateur de niveau instruction. Le principal inconvénient de cette approche est le temps de simulation très long. C'est un facteur important pour les méthodes de développement d'un code temps réel qui nécessite un nombre important d'itérations pour sa mise au point (respect des contraintes). Notre choix s'est donc porté sur une méthode basée sur l'addition des temps d'exécution des instructions générées par le compilateur appartenant à un même bloc de base [8]. Cette approche est appropriée à la classe de processeurs DSP concernée puisque ces derniers ne sont pas des processeurs superscalaires et ne possèdent pas de mémoire cache. Il existe cependant des instructions dont le nombre de cycles dépend des données. Ce problème se limite néanmoins aux instructions de branchements conditionnels. De plus, notre approche permet de lever toute ambiguïté sur ce type d'instructions puisque nous connaissons, après exécution de la séquence de test, le nombre de fois que le test est vrai ou faux.

3.3 Le processus d'estimation

La phase finale, ou « *back-end* », débute par la compilation du code C (annoté d'informations dynamiques) avec le compilateur du DSP cible.

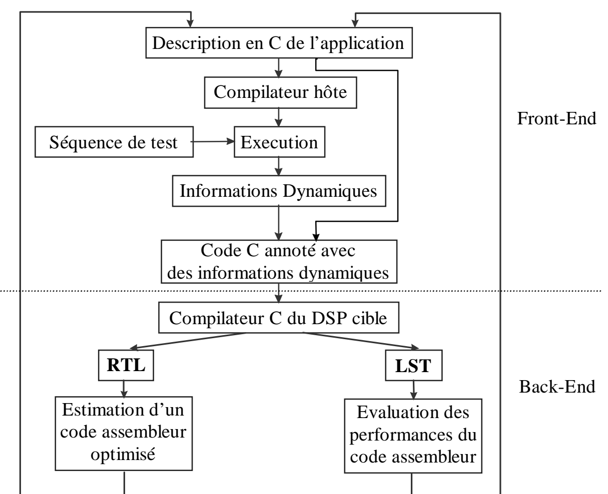

FIG.1 : Flot de l'estimateur VESTIM

3.3.1 Evaluation des performances du code assembleur généré

Le calcul des performances du code assembleur généré (LST) est effectué en utilisant directement les résultats de compilation. Nous additionnons les nombres de cycles des instructions assembleurs correspondants à chaque instruction C pour le calcul des c_i de chaque bloc de base. Une table construite à partir du manuel utilisateur décrivant le jeu d'instructions du DSP cible permet de déterminer un nombre de cycles associé à chaque instruction assembleur. Le calcul des performances par fonction est obtenu en multipliant le nombre d'exécutions de chaque bloc de base (x_i) par le nombre d'instructions assembleur par bloc de base.

3.3.2 Estimation d'un code assembleur optimisé

L'estimation d'un code assembleur optimisé opère sur une représentation intermédiaire des compilateurs C du projet GNU : la représentation RTL (Register Transfer Language). Afin de supporter une classe d'architecture DSP, nous adoptons un modèle générique interne du processeur. Le processeur cible est caractérisé par l'utilisation d'une description externe du processeur. Le OakDSPCore et le

PalmDSPCore sont les deux premiers processeurs DSP inclus dans notre estimateur. Les travaux relatifs à la méthode d'estimation d'un code assembleur optimisé ont déjà été présentés en détail et publiés dans [11].

3.4 Inconvénient d'une approche statistique

Plusieurs expérimentations, en particulier avec l'application G.728 [3][9], nous ont permis de montrer que les estimations d'un code optimisé sont très proches des performances obtenues lors d'un codage manuel. Néanmoins, ces estimations sont basées sur l'exécution de séquence de tests afin de déterminer le nombre d'exécution x_i de chaque instruction du code C. Pour les parties de code dont l'exécution ne dépend pas des valeurs des données en entrée (e.g. FFT), cette approche convient. Dans le cas contraire, les séquences de test ne garantissent pas une couverture de tous les chemins possibles du programme lors de son exécution et des erreurs d'estimation peuvent apparaître. De plus, certaines applications ne disposent pas de séquence de test comme la norme AC3 [2]. Dans ces cas, nous proposons d'éviter la construction laborieuse et difficile de séquences de test représentatives et d'étendre la méthode d'estimation par une approche théorique de type ILP (Integer Linear Programming) qui, par la formulation d'un ensemble de contraintes linéaires tirées de l'application, permet de déterminer les valeurs des x_i qui correspondent au pire cas.

4. Extensions de la méthode d'estimation par une approche théorique

Si la détermination des x_i ne posent pas de problème particulier dans le cas d'applications statiques, (i.e. des applications où toutes les caractéristiques des données, les traitements à effectuer et tous les comportements du système sont spécifiés, quantifiés et constants), cette détermination dans le cas dynamique est plus délicate. Le fait par exemple de rendre statique l'application en maximisant systématiquement les x_i de chaque bloc de base afin de caractériser le cas le plus défavorable [10], donc a priori la borne maximale, risque d'engendrer un pessimisme excessif. Par exemple, en prenant pour le calcul d'une FFT 1024 points, le nombre maximum d'itérations des trois boucles imbriquées typiques de cet algorithme, nous obtenons une estimation environ 500 fois supérieure au temps d'exécution réel mesuré.

L'objectif de l'approche ILP [12] est de déterminer une borne de performance et les x_i associés sans introduire de pessimisme excessif. La détermination du temps d'exécution dans le pire cas revient à maximiser l'équation linéaire définie en (1), ce qui se ramène à résoudre un ensemble de problèmes de programmation linéaire en nombres entiers (ILP). Les valeurs des x_i dépendent des contraintes structurelles et fonctionnelles inscrites dans le graphe de flots de contrôle (CFG) de l'application (le CFG de la figure 2 par exemple). Les contraintes *structurelles peuvent* être extraites directement du CFG [7]. Le nombre d'exécution x_i est égal à la fois à la somme de contrôle des flots entrants et la somme de contrôle des flots sortants de chaque bloc de base, comme le montre la Figure 2. Les d_i représentent le nombre de fois où le programme passe par les arc du graphe lors de son exécution. Les contraintes résultantes forment un ensemble de contraintes conjonctives, i.e. devant être satisfaites simultanément. Après avoir construit l'ensemble des contraintes structurelles, il est demandé à l'utilisateur de fournir des contraintes fonctionnelles, i.e. des informations qui dépendent des fonctionnalités du programme. Il s'agit de renseigner l'outil sur des relations entre différentes parties du programme qu'il n'est pas possible de déterminer automatiquement à partir du CFG. Les bornes de boucles sont définies dans la spécification des contraintes *fonctionnelles*. Ces dernières doivent permettre de trouver les chemins impossibles du graphe représentant le programme afin d'éviter un pessimisme excessif dans l'estimation des performances.

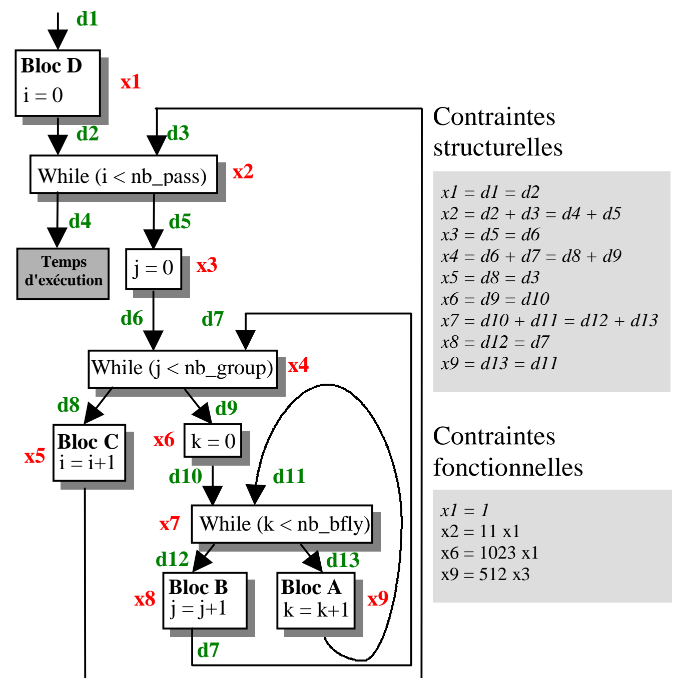

FIG.2 : CFG de la FFT et contraintes associées

Les contraintes fonctionnelles peuvent présenter des contraintes disjonctives, formant des ensembles mutuellement exclusifs de contraintes conjonctives. Chaque ensemble de contraintes fonctionnelles est alors combiné avec les contraintes structurelles. Parmi ces ensembles se trouve une solution satisfaisant toutes les contraintes de l'application. Dans la mesure où il est possible d'exprimer les contraintes sous une forme linéaire, il est alors aisé de déterminer la valeur maximale de l'expression obtenue, i.e. une borne de performance correspondant au cas le plus défavorable.

5. Résultats

La Table 1 montre les performances obtenues pour le OakDSPCore à partir d'un code assembleur généré par le compilateur C (LST), d'un code assembleur optimisé et écrit à la main (ASM), d'une estimation d'un code optimisé en utilisant l'outil VESTIM et une approche ILP. Les trois premières mesures sont obtenues à partir de l'exécution d'une même séquence de test. Les résultats montrent tout d'abord que le code généré par le compilateur C (LST) est très inefficace par rapport à un code assembleur écrit et optimisé

manuellement (ASM). La table 1 montre également que les estimations de performances fournies par VESTIM sont proches de celle mesurées pour un code optimisé (ASM). Comme la FFT et le bloc14 de G.728 sont des fonctions dont l'exécution ne dépend pas des valeurs des données en entrée (déterministes), les écarts de performance observés entre ces deux mesures pour ces deux fonctions proviennent uniquement des valeurs des c_i. Pour le bloc17, non déterministe, la différence de performance peut provenir à la fois des valeurs de c_i et des x_i.

Les résultats fournis en utilisant une approche ILP sont obtenus en considérant le CFG bâti à partir du code C. Les x_i sont déterminés grâce aux contraintes fonctionnelles fournies par l'utilisateur alors que les c_i proviennent d'une analyse du code assembleur optimisé. Pour des fonctions déterministes comme la FFT et le bloc14 de G.728, les différences de performance par rapport au code optimisé (ASM) proviennent uniquement d'imprécisions sur le calcul des c_i. Celles-ci sont dues aux différences qui apparaissent entre le CFG construit à partir du code C et celui obtenu à partir du code ASM. Notons cependant que si les contraintes fonctionnelles sont erronées ou incomplètes, des erreurs d'estimations peuvent apparaître même pour une fonction déterministe.

TAB.1 : Résultats comparatifs

Fonctions	LST	ASM	VESTIM	ILP
FFT 1024 points	543722 cyc (+310%)	**132336 cycles**	107699 cyc (-19%)	151820 cyc (+14,7%)
G.728 Bloc14	6,27 Mips (+200%)	**3,14 Mips**	3,08 Mips (-1,8%)	3,7 Mips (+17,8%)
G.728 bloc17	11,1 Mips (+57%)	**7,05 Mips**	6 ,37 Mips (-9,6%)	9,07 Mips (+28,7%)

Pour le bloc17, dont le contrôle dépend des données en entrée, la mesure fournie par l'approche ILP permet d'avoir une mesure de confiance par rapport aux résultats fournis par VESTIM. La performance de 9,07 Mips peut indiquer que la séquence de test utilisée en entrée de VESTIM ou pour la mesure de performance du code optimisé (ASM) ne couvre pas tous les cas possibles et en particulier le cas pire.

6. Conclusion

D'un point de vue industriel, les approches basées sur la programmation linéaire comme celle présentée dans cet article nous paraissent très contraignantes et difficilement envisageables pour l'estimation de performance d'une application complète. L'amélioration de la précision des estimations nécessite en effet l'ajout par l'utilisateur d'informations sur le comportement du programme. Dans le cas d'applications de taille importante le travail demandé à l'utilisateur est malheureusement très important. De plus, ces informations ne sont pas toujours une condition suffisante pour des analyses précises de performance. En effet, ces méthodes ne garantissent pas que toutes les relations entre différentes parties du programme sont exploitées. Néanmoins, ces approches sont une alternative intéressante dans le cas où il n'existe pas de séquence de test ou lorsque des parties de code dépendent des valeurs des données en entrée. Cependant, nous pensons que l'utilisation de ces méthodes ne doit concerner que les parties critiques ou non déterministes d'une application. Cette approche permet alors d'obtenir une mesure de confiance des résultats obtenus par l'exécution du code avec une séquence de tests. Nous pensons ainsi étendre à l'avenir notre outil d'estimation d'un solveur ILP du type de celui développé dans [4]. Les extensions en cours d'étude concernent également les méthodes d'estimation de consommation pour les processeurs DSP embarqués. VESTIM semble en effet un outil intéressant pour aider le programmeur dans l'optimisation du code C pour réduire la consommation.

Références

[1] B-S. Ovadia, G. Wertheizer et E. Briman. *Multiple and parallel execution units in digital signal processors.* ICSPAT, 13-16 septembre 1994, Toronto, pp 1491-1497.

[2] Advanced Television System Committee. *Digital Audio Compression Standard : AC-3.* Norme de compression de ATSC, document A/52, 20 décembre 1995.

[3] Recommendation G.728 : *Coding of speech at 16kbit/s using Low-Delay code excited linear prediction*. ITU, 1994.

[4] S. Malik et M. Martonosi. *Static Timing analysis of Embedded Software*. 34th DAC, 1997, pages 147-152.

[5] G. Araujo and al. - Challenges in Code Generation for Embedded Processors. Code Generation for Embedded Processors : 1st International Workshop, Edited by Peter Marwedel and Gert Goossens, pp. 48-64, Kluwer Academic Publishers, 1995.

[6] W. Ye, R. Ernst, Th. Benner and J. Henkel - *Fast Timing Analysis for Hardware-Software Co-Synthesis*, Proc. IEEE International Conference on Computer Design (ICCD), pp. 452-457, Cambridge, Massachusetts, USA, 1993.

[7] A.V. Aho, R. Sethi, J.D. Ullman – *Compilers: Principles, Techniques and Tools*, Addison-Wesley, 1986.

[8] C.Y. Park, A.C. Shaw – *Experiments With A Program Timing Tool Based On Source-Level Timing Schema*, The Journal of Real-Time Systems, vol. 1, n°2, pp. 160-176, September 1989.

[9] A. Pegatoquet, M. Auguin, E. Gresset and L. Bianco - *DSP Code Optimization using estimation metrics – A case study: G.728 on the OakDSPCore*. To appear in ICSPAT'99, 1-4 Novembre 1999, Orlando, Floride, USA.

[10] A.C. Shaw – *Reasoning about Time in Higher-Level Language Software*, IEEE Transactions on Software Engineering, vol. 15, n°7, pp.875-889, July 1989.

[11] A. Pegatoquet and al. - *Rapid development of optimized DSP code from a high level description Through software estimations*. 36th DAC, 21-25 Juin 1999, New Orleans, LA, USA.

[12] Y-T S. Li and S. Malik – *Performance Analysis of Embedded Software using Implicit Patrh Enumeration*. 32nd DAC, juin 1995, pages 456-461.