Estimation de la consommation d'un algorithme C intégrant l'architecture cible : une aide efficace pour la conception système

Johann Laurent, Nathalie Julien, Eric Senn, Eric Martin

Estimation de la consommation d'un algorithme C intégrant l'architecture cible : une aide efficace pour la conception système

Johann Laurent[1], Nathalie Julien[1], Eric Senn[1], Eric Martin[1]

[1]Laboratoire LESTER

Université de Bretagne Sud

Rue Saint Maude BP92116 56321 Lorient CEDEX

Prénom.nom@univ-ubs.fr

Résumé- Une méthode d'estimation de la consommation au niveau C pour processeurs VLIW a été développée. Cette méthode est basée sur une analyse fonctionnelle de la consommation du processeur dont le résultat est un modèle de consommation de la cible. Ce modèle décrit le comportement en puissance du processeur à l'aide de paramètres algorithmiques et de configuration. Certains de ces paramètres peuvent être directement prédits à partir de l'algorithme C, sans compilation, avec des connaissances simples sur l'architecture cible et le compilateur. Nous présentons comment notre méthode permet, dès le niveau C et par le développement de modèles de prédictions adéquats, de tenir compte des fortes variations de consommation du code final, soit dues aux différentes options de compilation, soit aux modifications du code source. L'erreur maximale d'estimation est de 10% par rapport aux mesures pour des algorithmes classiques de traitement du signal et de l'image sur un processeur TMS320C6201.

Abstract *– A consumption estimation method at the C level for VLIW processors has been developed. This method is based on a functional level power analysis of which the result is a consumption model of the target. This model represents the processor power behavior by using algorithmic and configuration parameters. Some of these parameters can be predicted directly at the C algorithm level, without compilation, only with simple assumptions about the architecture and the compiler. We present how our method allows, directly at the C level and with some prediction models, to take into account the consumption variations, which are due to the compilation optimizations. The maximum error between our estimations and the measures is 10% for classical DPS algorithms executed on a TMS320C6201.*

1. Introduction

Parmi les différents critères d'évaluation, la consommation s'est dernièrement imposée comme un paramètre de conception aussi critique que la vitesse et la surface. Pour les systèmes embarqués, l'augmentation des capacités de calcul et des fréquences de fonctionnement des circuits entraîne une surconsommation qui pénalise aussi bien l'autonomie du circuit que son coût ou sa fiabilité. Afin de maîtriser la consommation finale d'un système complet, il devient donc indispensable de l'étudier dès la phase de conception pour chaque application car mesurer la consommation après l'implémentation est contraignant et il n'est plus possible, à ce stade, de revenir sur les choix de conception. Mais il ne suffit pas de vérifier si les contraintes de l'application sont respectées ou non, il est également nécessaire de comparer plusieurs solutions entre elles. Il faut donc disposer d'outils d'estimation de haut-niveau qui renvoient au concepteur des informations sur la qualité de sa solution pour l'application visée et le guident dans l'espace de conception.

Or, pour qu'une estimation soit fiable, elle doit comporter les caractéristiques suivantes :

- elle doit être précise malgré le niveau élevé d'abstraction, et de préférence validée par rapport à des mesures.
- elle doit être rapide c'est à dire que le temps de retour d'information pour le concepteur ne doit pas excéder quelques minutes.
- elle doit être complète en prenant en compte tous les différentes sources de consommation notamment la consommation statique.
- elle doit être générale en pouvant être effectuée sur différentes cibles et ce indépendamment des outils constructeurs.

Il existe des méthodes d'estimation de la consommation basée sur des simulations au niveau cycle comme dans Wattch ou SimplePower [1,2]. Cependant, ces méthodes, d'une part demandent un temps de calcul important, et d'autre part, reposent sur un modèle très détaillé de description de l'architecture, qui est souvent indisponible lorsqu'il s'agit de processeurs du commerce. Une autre approche classique est d'évaluer la consommation avec une méthode d'analyse au niveau instruction [3]. Cette méthode repose sur les mesures de courant pour chaque instruction et chaque couple d'instructions successives. Sa principale limitation est donc le nombre de mesures nécessaires afin d'élaborer le modèle d'un processeur d'architecture complexe (ex : 1176 mesures pour un DSP 56K[4]). Certaines études récentes ont ajouté une approche fonctionnelle à cette approche au niveau instruction [5,6] mais peu de travaux concernent les processeurs VLIW [7]. Toutes ces études fournissent des estimations au niveau assembleur avec des précisions variant de 4% à 10% suivant la complexité du modèle considéré.

Mener une estimation directement au niveau du code C sans compilation présente plusieurs avantages. Afin de déter-

miner l'architecture la plus adaptée à une application, différentes cibles peuvent être testées sans acquérir ni le processeur ni son environnement de développement, sous réserve que le modèle de consommation de la cible soit disponible. On peut également tester des parties de programmes, des fonctions, ou des programmes non finalisés, qui ne pourraient pas être compilés mais qui représentent des parties critiques pour les caractéristiques finales du système. Mais pour obtenir une estimation algorithmique précise, il est impératif de tenir compte non seulement de la consommation inhérente à l'algorithme (quantifiée par le nombre d'opérations et le nombre d'accès mémoire) mais également de l'architecture ciblée [8]. En effet, un même algorithme, destiné à un processeur permettant du parallélisme ou non, fournira des résultats en terme de performance et de consommation très différents.

Nous nous proposons d'illustrer ce problème et de fournir une solution grâce à une méthode originale d'estimation de la consommation d'un algorithme au niveau C sans compilation mais intégrant les caractéristiques de la cible via des modèles de prédiction anticipant le résultat et l'efficacité du compilateur. Le paragraphe 2 présentera l'élaboration du modèle de consommation du processeur. Puis, la méthode d'estimation au niveau C est présentée dans le paragraphe 3 et enfin des applications sur des algorithmes de traitement du signal seront présentées dans le paragraphe 4 pour montrer la validité de notre approche et son utilité dans la conception système.

2. Elaboration du modèle d'un processeur

Avant de réaliser l'estimation de consommation proprement dite, il faut disposer de modèles de processeur en bibliothèque. L'élaboration du modèle de puissance, réalisée une seule fois par cible, est basée sur l'analyse fonctionnelle de l'architecture [9]. Cette analyse, indépendante du niveau d'abstraction (assembleur ou C), permet de déterminer les phénomènes prépondérants du point de vue de la consommation. De plus, notre méthode prend en compte toutes les fonctions du processeur que ce soit le parallélisme, le contrôle du pipeline, les unités de traitement, les mémoires internes ainsi que les défauts de cache.

Le modèle de puissance représente un ensemble de lois de consommation exprimant le courant du cœur du DSP I_{TOTAL} en fonction de paramètres d'entrée du modèle, les paramètres algorithmiques et les paramètres de configuration. Les coefficients de ces lois sont obtenus à partir d'un nombre restreint de mesures caractérisant la cible.

Nous présentons en Figure 1 le modèle obtenu pour le DSP TMS320C6201 de Texas Instruments. Ce type de processeur a été choisi pour la complexité de son architecture, qui interdit d'appliquer une méthode au niveau instruction (celle-ci nécessiterait au minimum 71^8 mesures). En effet, il possède un pipeline profond (11 étages), un jeu d'instructions VLIW (256 bits), une possibilité d'avoir jusqu'à 8 instructions parallèles, deux mémoires internes (données et programme) ainsi qu'un EMIF (External Memory InterFace) pour les accès aux mémoires externes [10].

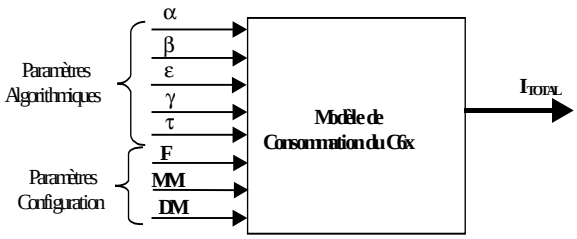

Fig.1. Modèle de consommation du C6201

Les paramètres algorithmiques représentent le taux d'activité entre chaque bloc fonctionnel du DSP et leurs valeurs dépendent de l'algorithme à estimer : α représente *le taux de parallélisme* du programme, β *le taux d'utilisation des unités de traitement*. Le paramètre ε concerne *le taux d'accès via le DMA* ; ces accès sont décrits par le programmeur directement en langage assembleur. Le paramètre γ représente *le taux de défaut de cache* et τ *le taux d'accès des données en mémoire externe*.

Les paramètres de configuration, définis par le concepteur, sont *la fréquence d'horloge F* pouvant atteindre 200 MHz, *le mode mémoire MM* définissant le mode d'utilisation de la mémoire de programme interne (mapped, cache, freeze ou bypass) [9] et *le placement des données DM* indiquant si les données sont placées en mémoire interne et dans quel banc mémoire.

3. Méthode d'estimation

Le processus d'estimation consiste à extraire du code source les valeurs des paramètres qui servent d'entrée au modèle. Or, les paramètres de configuration sont définis par l'utilisateur en même temps que l'application. Il nous faut donc déterminer les paramètres algorithmiques, et ce, sans compiler le code C. Ce processus d'estimation, très rapide, repose sur un profiling statique du code C qui s'applique aux boucles du programme. Il faut que l'algorithme soit constitué de boucles de taille suffisante pour pouvoir négliger tout ce qui n'est pas inclus dans les boucles, condition vérifiée dans la plupart des algorithmes de traitement du signal et de l'image. Les paramètres algorithmiques sont estimés en utilisant un modèle de prédiction, qui est en fait une hypothèse sur le résultat de la compilation et l'efficacité du compilateur à utiliser l'architecture ciblée. Il suffit ensuite d'utiliser les lois de consommation établies pour l'architecture cible pour connaître la consommation de l'application. Plusieurs cibles peuvent donc être évaluées si plusieurs modèles de processeurs sont disponibles en bibliothèques.

Le C6x charge 8 instructions simultanément, formant un paquet de chargement (FP). Toutes les instructions exécutées en parallèle forment un paquet d'exécution (EP) et chaque FP peut être constitué de plusieurs EP. Les paramètres α et β sont alors calculés de la façon suivante :

$$\alpha = \frac{NFP}{NEP}(1 - PSR)$$
$$\beta = \frac{1}{NPU_{MAX}} \frac{NPU}{NEP}(1 - PSR) \qquad (1)$$

où *PSR* représente *le taux de rupture de pipeline*, *NFP* et *NEP* représentent respectivement *le nombre moyen de FP et de EP*, *NPU* représente *le nombre moyen d'unités fonctionnelles utilisées* (toute opération sauf l'instruction NOP) et NPU_{MAX} représente *le nombre maximum d'unités fonctionnelles* pouvant être utilisées (NPU_{MAX} = 8 pour le C6x).

Nous pouvons donc estimer les paramètres α et β, à partir du source C, en prédisant le nombre de paquets de chargement, d'exécution et le nombre d'unités de traitement utilisés par l'application. De même, nous pouvons prédire *le taux d'accès en mémoire externe* τ grâce au *paramètre de configuration DM*. *Le taux de rupture de pipeline PSR* est obtenu par le nombre de cycles où le pipeline est gelé NPS sur le nombre de cycles total pour le programme NTC. Or, NPS est la somme des cycles pour chaque cause de rupture de pipeline : les accès en mémoire externe (quantifiés par τ), les conflits d'accès à un banc mémoire (déterminés par le paramètre DM), les défauts de cache (paramètre γ) ou les aléas de contrôle (négligés pour notre classe d'applications).

Comme indiqué précédemment, le paramètre ε est connu du programmeur quand il décrit les différents accès DMA de son application. Pour déterminer exactement γ, il faut réaliser une trace du programme. Cependant, de nombreux processeurs actuels ont une taille de mémoire interne de programme conséquente (64Ko ici) permettant de contenir la plupart des applications qui nous intéresse. Dans le cas contraire, nous fournissons alors une 'carte de consommation' représentant les variations de l'application en fonction du taux de défaut de cache γ.

Nous avons déterminé 4 modèles de prédiction basés sur une connaissance minimale de l'architecture cible et du compilateur :

- Le modèle séquentiel SEQ suppose que toutes les instructions sont traitées séquentiellement.
- Le modèle maximum MAX considère que le compilateur exploite entièrement les possibilités de l'architecture (par exemple le parallélisme est maximum) ; ce modèle fournit une borne maximum pour la consommation du processeur.
- Pour le modèle minimum MIN, aucun load et/ou store ne peuvent être en parallèle. Par contre, les opérations de calculs peuvent être effectuées en parallèle.
- Le modèle DATA suppose que le compilateur exploite toutes les possibilités architecturales mais les instructions load et/ou store en parallèle sont limitées à des données différentes.

Lorsque l'application est constituée de plusieurs fonctions et/ou de boucles, les valeurs locales des paramètres sont prédites pour chaque fonction et boucle. Les paramètres globaux de l'algorithme sont ensuite obtenus en calculant une moyenne pondérée de ces valeurs locales.

4. Applications

Dans de précédents travaux, nous avons déjà démontré la validité de notre approche puisque l'erreur ente la mesure et l'estimation se situe au maximum à 8% pour un ensemble d'applications en traitement du signal et de l'image [9]. Ici, nous nous proposons, dans un premier temps, d'illustrer l'influence des options de compilation sur la consommation et comment notre méthode peut prévoir les variations de consommation du code final. Dans un deuxième temps, nous illustrerons l'impact des modifications algorithmiques sur la consommation et comment notre méthode peut suivre précisément les variations algorithmiques afin d'en rendre compte au programmeur.

4.1 Influence des options de compilation

Le problème principal lorsque l'on veut estimer au niveau C est la prise en compte des options de compilation qui vont agir sur les performance du code final. Habituellement, deux possibilités sont offertes : l'optimisation en code ou l'optimisation en performance. En réalité, pour chacune de ces deux options, l'utilisateur choisit ensuite souvent le niveau d'optimisation le plus élevé.

Le tableau 1 présente des mesures et des estimations pour une application MPEG, dont le code C est constitué de 11 fonctions soit 2267 lignes de codes. La fréquence d'horloge est de F = 200 MHz, le mode mémoire est le mode Mapped ce qui correspond à un programme en interne. Les données sont également placées en interne. Nous fournissons les mesures de temps d'exécution T_{exe}, de puissance P et d'énergie E obtenues pour trois cas de figures. Tout d'abord sans option de compilation ; dans ce cas, le code reste purement séquentiel. Pour l'optimisation en performance, le maximum de parallélisme est obtenu, en utilisant le déroulage de boucle et le pipeline logiciel ; évidemment la puissance du processeur augmente puisque ses ressources sont mieux utilisées mais l'énergie est beaucoup plus faible. Enfin l'optimisation en code où aucun déroulage de boucle n'a lieu mais où on trouve encore du parallélisme au sein des boucles ; la puissance est plus faible et l'énergie plus importante que pour l'optimisation en performance.

Tableau 1 : Influence des options de compilation pour l'application MPEG

	Mesures			**Estimations Puissance (W)**			
	Texe (µs)	**P (w)**	**E (µJ)**	**SEQ**	**MAX**	**MIN**	**DATA**
	Sans option						
MPEG	*580*	*2.71*	*1.57*	***2.71***	*6.47*	*3.96*	*5.42*
	Performance						
	40.37	*5.83*	*235.08*	*2.71*	*6.47*	*3.96*	***5.42***
	Code						
	178.33	*3.02*	*539*	***2.71***	*6.47*	*3.96*	*5.42*

Les résultats des estimations au niveau C sont fournis pour les 4 modèles de prédiction définis précédemment. Le résultat obtenu par le modèle séquentiel SEQ est parfaitement

identique à la mesure sans option de compilation. De plus, ce modèle permet d'estimer la consommation pour l'optimisation en code avec une erreur de 10,3%. En réalité, dans ce dernier cas, il sous-estime la consommation puisque l'on suppose qu'aucun parallélisme n'est obtenu, ce qui n'est pas tout à fait exact. Pour l'optimisation en performance, le modèle DATA fournit une estimation précise à 7% près. La précision de ces résultats est comparable à ceux obtenus avec une méthode au niveau assembleur.

Donc, non seulement nos modèles permettent d'estimer précisément la puissance dès le niveau C, mais ils fournissent également des bornes : l'optimisation en performance est bornée par les modèles MIN et MAX alors que l'optimisation en code est bornée par les modèle SEQ et MIN. On pourrait envisager de développer d'autres modèles de prédiction, notamment pour une évaluation plus précise de l'optimisation en code. Cependant, nos efforts se sont surtout portés sur l'optimisation en performance car c'est celle qui est le plus efficace pour l'énergie. De même, si d'autres compilateurs étaient utilisés, comme par exemple des compilateurs spécifiques à l'optimisation de consommation, de nouveaux modèles de prédiction pourraient aisément être introduits.

4.2 Impact de la consommation d'un algorithme

Le tableau 2 présente les résultats pour deux versions d'un algorithme de transformation en ondelettes avec 3 niveaux de résolution sur une image 512*512 compilé avec une optimisation en performance. La fréquence d'horloge est de 200 MHz, le programme est en interne mais les données sont en externe. Le premier algorithme (Algo1) effectue, de façon classique, un réarrangement de l'image à la fin de chaque traitement pour le niveau considéré ; or, dans ce cas, 85% du temps de calcul est alors occupé par ce réarrangement. Nous avons donc légèrement modifié les boucles de cet algorithme afin d'éviter ce réarrangement. La conséquence est donc de limiter le nombre d'accès en mémoire externe et donc de fortement influencer les performances et la consommation. Cela se traduit par une réduction de la puissance de 44% et une diminution d'énergie de 94%! L'impact d'une telle modification algorithmique prouve la nécessité de disposer d'outils d'estimation de haut niveau rapide et fiables.

Tableau 2 Mesures et estimations des deux versions de la transformée en ondelettes

Algo	Mesures			Estimations	
	Texe (ms)	P (W)	E (J)	P (W)	Δ (%)
Algo-1	5700	4.53	25.82	4.32	**-4.6**
Algo-2	577.8	2.55	1.47	2.61	**+2.4**

Les estimations, obtenues avec le modèle de prédiction DATA, suivent précisément les variations de puissance des algorithmes. Le programmeur peut donc valider rapidement les améliorations de ses programmes, et les tester sur plusieurs cibles. Dans la version finale du papier, nous ajouterons des estimations et des mesures pour un autre processeur, le C55, que nous comparerons avec les performances du C6x.

5. Conclusion

Nous avons présenté une méthode d'estimation de la puissance d'un code au niveau C pour des processeurs VLIW. Pour réaliser cette estimation, il n'est pas nécessaire d'avoir une connaissance approfondie de l'architecture cible mais seulement une connaissance fonctionnelle du compilateur et de l'architecture cible. Elle permet, par le développement de modèles de prédiction adéquats, de rendre compte des fortes variations de consommation du code final, soit dues aux différentes options de compilation, soit aux modifications du code source. La précision de cette méthode en fait un outil efficace d'aide à la conception système. Les travaux en cours portent sur l'application de cette méthode sur d'autres processeurs. Les travaux futurs seront le développement d'un outil automatique accessible en ligne.

6. Bibliographie

[1] D. Brooks, V. Tiwari, M. Martonosi "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations" in Proc. ISCA (2000).

[2] W. Ye, N. Vijaykrishnan, M. Kandemir, M.J. Irwin "The Design and Use of SimplePower: A Cycle Accurate Energy Estimation Tool" In Proc. Design Automation Conférence (2000).

[3] V. Tiwari, S. Malik, A. Wolfe "Power analysis of embedded software: a first step towards software power minimization." *IEEE Transaction on. VLSI Systems, Décember 1994.*

[4] B. Klass, D.E. Thomas, H. Schmit, D.F. Nagle "Modeling Inter-Instruction Energy Effects in a Digital Signal Processor" *Power Driven Microarchitecture Workshop ISCA, 1998.*

[5] S. Steinke, M. Knauer, L. Wehmeyer, P. Marwedel "An Accurate and Fine Grain Instruction-Level Energy Model Supporting Software Optimizations" in Proc. PATMOS (2001)

[6] G. Qu, N. Kawabe, K. Usami, M. Potkonjak "Function-Level Power Estimation Methodology for Microprocessors" in Proc. Design Automation Conference (2000)

[7] L. Benini, D. Bruni, M. Chinosi, C. Silvano, V. Zaccaria, R. Zafalon "A Power Modeling and Estimation Framework for VLIW-based Embedded Systems," in *Proc. Int. Workshop on Power And Timing Modeling, Optimization and Simulation PATMOS,* Sept. 2001, pp. 2.1.1-2.1.10.

[8] J. Rabaey, M. Pedram *"Low Power Design Methodologies"* Kluwer Academic Publishers 1996

[9] N. Julien, J. Laurent, E. Senn, E. Martin " *Power Estimation of a C Algorithm Based on the Functional-Level Power Analysis of a DSP* " Lecture Notes in Computer Science, Springer Verlag, April 2002, N°2327.

[10] *Documentation Texas Instruments* Technical Brief (SPRU 197D) février 1999.