Etude de la consommation de plates-formes multiprocesseurs

Johann Laurent, Eric Senn, Nathalie Julien, Eric Martin

Etude de la consommation de plates-formes multiprocesseurs

Johann Laurent, Eric Senn, Nathalie Julien, Eric Martin
Laboratoire LESTER- Université de Bretagne Sud, Centre de recherche 2 rue Saint Maudé
BP92116 56321 LORIENT CEDEX France
johann.laurent@univ-ubs.fr, http://lester.univ-ubs.fr :8080

Résumé
Aujourd'hui, le développement d'applications multimédia embarquées engendre un besoin de puissance de calcul toujours plus important. Face à cette augmentation, la solution monoprocesseur atteint rapidement ses limites. La solution alors envisagée par les concepteurs est l'emploi de plates formes multiprocesseurs. S'il est évident que cette solution entraîne un gain en terme de puissance de calcul, qu'en est-il pour la consommation de puissance et d'énergie ? Nous verrons ici, en utilisant un exemple concret, qu'il n'est pas trivial de déterminer si une solution multiprocesseur est plus intéressante qu'une solution monoprocesseur.
Mots Clés: Consommation, Multiprocesseur, Estimation

1. Introduction

Sur le marché actuel, nous voyons émerger de plus en plus de systèmes nomades (PDA, SmartPhone etc…) ayant un spectre d'application toujours plus vaste. Ces nouveaux systèmes sont caractérisés par un besoin important de ressources de calcul et de mémorisation afin d'atteindre les performances désirées [1]. De plus, la flexibilité devient un critère important pour ces systèmes ce qui a pour effet l'emploi de processeurs ou de composants reconfigurables [2].
Dans le cas d'utilisation de processeurs, deux solutions peuvent être envisagées pour atteindre les performances requises : l'emploi de co-processeurs spécialisés ou l'emploi d'architectures multiprocesseurs. La solution co-processeur a comme avantage majeur de meilleures performances en terme de temps d'exécution et de consommation. Son principal défaut est un manque de flexibilité par rapport à l'approche multiprocesseurs.
Dans cet article, nous ne traiterons que le cas de plate-formes multiprocesseurs et nous démontrerons qu'il n'est pas trivial de déterminer quelle doit être l'architecture à utiliser afin d'en optimiser le temps d'exécution et la consommation. Cette démonstration se fera en utilisant un exemple concret d'application multimédia : un encodeur MPEG-2.
Pour pouvoir réaliser cette étude, nous devons disposer d'une méthode et d'un outil nous permettant d'obtenir rapidement une estimation du temps d'exécution et de la consommation engendrés par l'exécution d'une application sur une architecture cible.

2. Modèles de consommation

La première étape, pour mener à bien cette étude, consiste à développer des modèles de consommation pour différentes architectures pouvant être utilisées sur la plate-forme multiprocesseurs. Nous devons donc employer une méthodologie de modélisation nous permettant d'obtenir rapidement ces modèles. Dans le cadre de cette étude nous emploierons la méthodologie FLPA (Functional Level Power Analysis) dévéloppée au LESTER ; en effet, cette méthodologie permet d'obtenir des modèles simples (même pour des architectures complexes) et de réaliser des estimations de bonnes précisions [3].

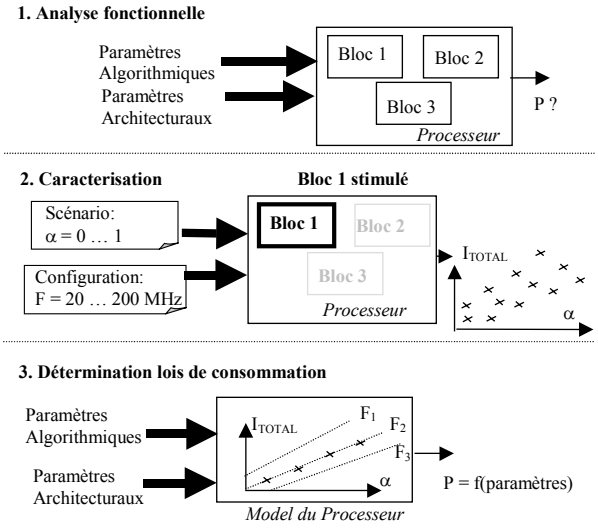

Fig.1 La méthodologie FLPA

Ces modèles de consommation utilisent deux types de paramètres : des paramètres architecturaux (mode mémoire, fréquence d'horloge…) ainsi que des paramètres algorithmiques (taux de parallélisme, taux de défauts de cache…).
Cette méthode possède un outil automatique associé, SoftExplorer ; il nous sera alors aisé d'étudier différents scénarii afin de déterminer le meilleur compromis temps d'exécution/consommation. En effet, cet outil permet à l'utilisateur d'estimer la consommation de puissance et d'énergie de son

application soit directement à partir du code C soit à partir du code assembleur généré par le compilateur. Il permet également d'obtenir une prédiction du temps d'exécution de l'application [4].

SoftExplorer possède, à l'heure actuelle, une librairie composée de 5 modèles de processeurs :

- Modèle du TMS320C6201 (VLIW virgule fixe)
- Modèle du TMS320C6701 (VLIW virgule flottante)
- Modèle du TMS320C6416 (VLIW virgule fixe)
- Modèle du TMS320VC5510 (Low Power virgule fixe)
- Modèle de l'ARM7TDMI (processeur général).

De nouveaux modèles sont, actuellement, en développement :

- Modèle de l'ARM920T (processeur général)
- Modèle du PowerPC (processeur général superscalaire)
- Modèle du Xscale (processeur général faible consommation)
- Modèle de l'OMAP (SoC multiprocesseur ARM9 + C55).

Une fois la totalité de ces modèles de consommation obtenue, nous aurons alors balayé la majorité des types d'architectures de processeurs possibles. Nous aurons ainsi démontré la généralité de la méthode de modélisation FLPA.

Il nous resterait toutefois à appliquer notre méthode sur des processeurs paramétrables et/ou reconfigurables. Ces processeurs ne diffèrent pas des autres d'un point de vue architectural mais, offrent la possibilité de faire varier statiquement ou dynamiquement leurs chemins de données et/ou leur nombre d'unités de traitement. Ceci est le cas, par exemple, sur le processeur MACGIC développé au CSEM (Centre Suisse d'Electronique et de Micro-électronique) et sur le processeur DART développé au sein de l'équipe R2D2 de l'ENSSAT à Lannion.

3. Méthode et contexte de l'étude

La plate-forme multiprocesseur virtuelle utilisée permet, au maximum, l'utilisation de 4 processeurs de traitement du signal en parallèle. Nous supposerons que tous les processeurs utilisés sur la plate forme sont homogènes (i.e. utilisation de 4 processeurs identiques).

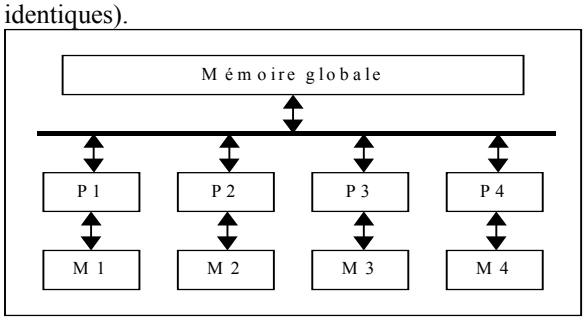

Fig2. Plate-forme multiprocesseur virtuelle

La plate-forme multiprocesseur virtuelle utilisée est composée de 5 types d'éléments différents :

- Processeurs hétérogènes P_i (dans notre cas ce sont des DSP).
- Mémoires locales M_i de type SRAM.
- Bus privés entre le processeur et sa mémoire locale associée.
- Mémoire globale de type SBRAM utilisée pour réaliser la communication ainsi que le transfert des données vers les processeurs.
- Bus partagé servant à la communication entre la mémoire globale et les processeurs.

L'application utilisée dans le cadre de cette étude sera un encodeur MPEG permettant la compression d'images animées au format MPEG-2. Nous commencerons par étudier la fonction d'estimation de mouvements de cette application puisque c'est elle qui requiert le plus de puissance de calcul. De plus, cette fonction est relativement facile à paralléliser afin de permettre l'utilisation de plusieurs processeurs. En effet, la division de la taille de l'image par deux (en hauteur et en largeur) permet l'utilisation de 4 processeurs afin de réaliser l'estimation de mouvements sur l'ensemble de l'image (cf. figure 3).

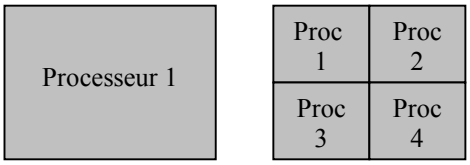

Fig3. Découpage du traitement de l'image

Dans le premier cas, un seul processeur s'occupera de réaliser l'estimation de mouvement sur l'ensemble de l'image. Les résultats obtenus dans ce cas nous servirons de résultats de référence. Dans le deuxième cas, chaque processeur s'occupera de réaliser l'estimation de mouvement sur un quart de l'image globale.

La puissance et l'énergie consommée par les processeurs seront calculées en utilisant les équations suivantes :

- ***cas monoprocesseur***

$$P = P_{P1} \quad (1)$$

$$E = P_{P1} \times T_{exeP1} \quad (2)$$

P_{P1} représente la consommation de puissance du processeur 1 et T_{exeP1} le temps d'exécution pour le processeur 1.

- ***Cas multiprocesseurs***

$$P = \sum_{i=0}^{n} P_{Pi} + \sum_{i=0}^{n} P_{idle} \quad (3)$$

$$E = \sum_{i=0}^{n} (P_{Pi} \times T_{exePi}) + \sum_{i=0}^{n} (P_{idle} \times T_{idle}) \quad (4)$$

P_{Pi} représente la consommation de puissance du processeur Pi et T_{exePi} le temps d'exécution pour le

processeur Pi. P_{idle} représente la consommation de puissance lorsque le processeur est en veille et T_{idle} le temps de mise en veille.

Afin de déterminer la consommation de puissance et d'énergie globale du système, nous devrons ajouter à la consommation des processeurs la consommation des différents bus et des différentes mémoires. Pour notre étude, nous considérerons que seule la mémoire globale est utilisée et que l'ensemble des données utiles aux calculs pourra être stocké dans la mémoire interne du ou des processeurs. Les consommations de puissance et d'énergie du système seront donc calculées en utilisant les équations suivantes :

- ***Cas système monoprocesseur***

$$P = P_{P1} + P_{Mem} + P_{MemIdle} \quad (5)$$

$$E = P_{P1} \times T_{exeP1} + P_{Mem} \times T_{MU} + P_{MemIdle} \times T_{MIdle} \quad (6)$$

P_{P1} représente la consommation de puissance du processeur 1 et T_{exeP1} le temps d'exécution pour le processeur 1. P_{Mem} représente la consommation de puissance de la mémoire globale lorsqu'elle est utilisée et $P_{MemIdle}$ la puissance lorsqu'elle est en veille. T_{MU} et T_{MIdle} représentent respectivement le temps d'utilisation et le temps de veille de la mémoire globale.

- ***Cas système multiprocesseurs***

$$P = \sum_{i=0}^{n} P_{Pi} + \sum_{i=0}^{n} P_{idle} + P_{Mem} + P_{MemIdle} \quad (7)$$

$$E = \sum_{i=0}^{n} (P_{Pi} \times T_{exePi}) + \sum_{i=0}^{n} (P_{idle} \times T_{idle}) + (P_{Mem} \times T_{MU}) + (P_{MemIdle} \times T_{MIdle}) \quad (8)$$

P_{Pi} représente la consommation de puissance du processeur Pi et T_{exePi} le temps d'exécution pour le processeur Pi. P_{idle} et T_{idle} représentent respectivement la consommation de puissance lorsque le processeur est en veille et le temps de mise en veille. P_{Mem} représente la consommation de puissance de la mémoire globale lorsqu'elle est utilisée et $P_{MemIdle}$ la puissance lorsqu'elle est en veille. T_{MU} et T_{MIdle} représentent respectivement le temps d'utilisation et le temps de veille de la mémoire globale.

Du fait de la virtualité de notre plate-forme, nous ne pourrons prendre en compte le coût en consommation du bus partagé entre la mémoire globale et le ou les processeurs. Nous fournirons donc une sous estimation de la consommation réelle. Toutefois cette étude n'est réalisée que pour mettre en exergue le fait qu'il n'est pas trivial de définir la meilleure architecture à utilisée dans un MPSoC (Multi Processor System on Chip). Donc, en considérant que ce coût énergétique est du même ordre de grandeur dans nos deux cas, une comparaison relative est possible.

4. Présentation et analyse des résultats

Dans la partie résultats, nous verrons quel est l'impact de l'architecture sur la consommation pour une plate-forme multiprocesseurs. En effet, le choix d'une architecture complexe engendre à la fois une augmentation la consommation de puissance et une diminution de la consommation d'énergie. Un compromis optimum entre la complexité de l'architecture et le couple consommation de puissance/énergie peut être dégagé afin de répondre au mieux aux contraintes de l'application.

4.1 Résultats de références : cas monoprocesseur

Dans cette première expérience, l'estimation de mouvements est réalisée par un seul processeur. Les résultats d'estimation de la consommation de puissance et d'énergie sont obtenus en utilisant l'outil d'estimation SoftExplorer (version 5.0). Les estimations seront réalisées en utilisant directement le code C fourni dans MediaBench. Dans un premier temps, le processeur utilisé sera le TMS320C6201 de TI. Les contraintes de mapping mémoire prendront en compte le fait que ce processeur possède 2 bancs mémoire interne pour les données ; chaque structure de données successive sera placée dans des bancs différents. Ce mapping mémoire n'est pas forcément le plus adéquate en terme de conflits mémoire mais sera utilisé pour toutes les expériences. Le nombre d'itération des boucles dynamiques a été déterminé par profiling dynamique en utilisant code Composer Studio (outil de développement de TI). La probabilité de passage dans les structures de contrôle sera considérée comme équiprobable (50% dans chacune des branches).
La taille de l'image traitée par l'estimateur de mouvements varie de 64x64 pixels jusqu'à 1024x1024 pixels. Les différents résultats sont présentés dans le tableau ci-dessous.

Texe (ms)	I (mA)	P (mW)	Pmax (mW)	E (mJ)
64x64 pixels				
0.132	1889.4	4723.5	7385	0.622
128x128 pixels				
0.259	1886.4	4716	7385	1.22
256x256 pixel				
0.508	1883	4708	7385	2.39
512x512 pixels				
1.007	1882	4705	7385	4.74
1024x1024 pixels				
2.015	1882	4705	7385	9.479

Table 1. Résultats dans le cas monoprocesseur

Au vue des ces résultats, nous pouvons constater que malgré l'augmentation de la taille d'image, le courant, la puissance moyenne et la puissance maximum sont constant. Ceci se justifie par le fait que le degré de

parallélisme du code n'est pas modifié lorsque la taille d'image augmente. Or, comme la consommation de puissance est fonction du nombre d'unités de traitement utilisées en parallèle, il est normale que ces consommations soient constantes si ce taux n'évolue pas. En revanche si la taille de l'image devient très petite alors, il se peut que ces consommations diminuent. En effet, si la taille de l'image est trop faible alors le compilateur ne pourra plus obtenir le parallélisme optimum (dû au déroulage des boucles de calcul) alors le degré de parallélisme diminuera donc la puissance diminuera également et le temps d'exécution augmentera.
En revanche, nous remarquons que plus la taille d'image augmente et plus le temps d'exécution et donc l'énergie augmente (puisque la puissance est constante). En effet, si la taille d'image augmente d'un facteur 16 (64x64 à 1024x1024 pixels) alors le temps d'exécution et l'énergie augmentent également d'environ un facteur 16.

4.2 Résultats : cas multiprocesseur

Dans le cas multiprocesseur, l'estimateur de mouvement est réalisé non plus avec un seul processeur mais en en utilisant 4. Pour cette étude, nous utiliserons le même contexte d'étude à savoir l'utilisation de SoftExplorer pour réaliser les estimations et l'utilisation de 4 DSP TMS320C6201. Le mapping mémoire restera également le même afin d'éviter de biaiser les comparaisons.
Avant même de faire l'expérience, nous pouvons déjà, a priori, dégager un certain nombre de résultats.
Premièrement, le fait d'utiliser 4 processeurs au lieu d'un va engendrer une augmentation de la puissance consommée d'un rapport 4 si le code généré par le compilateur garde le même degré de parallélisme. En effet, nous avons vu précédemment (c.f. paragraphe 4.1) que la taille de l'image n'influait pas sur les consommations de puissance.
Deuxièmement, le temps d'exécution va lui aussi diminuer du fait de l'utilisation de 4 processeurs. Nous pouvons nous attendre à une réduction de ce temps d'un facteur 4.
Le tableau 2 présente les résultats pour différentes tailles d'image (de 64x64 à 1024x1024 pixels).

Texe (ms)	I (mA)	P (mW)	Pmax (mW)	E (mJ)
64x64 pixels				
0.039	7721	19303	29540	0.753
128x128 pixels				
0.071	7668	19168	29540	1.36
256x256 pixel				
0.135	7638	19096	29540	2.584
512x512 pixels				
0.265	7624	19060	29540	5.044
1024x1024 pixels				
0.525	7620	19048	29540	10

Table 2. Résultats dans le cas multiprocesseur

A la vue de ces résultats nous remarquons que les puissances moyenne et maximum ont bien été augmentées d'un facteur 4 comme nous pouvions l'imaginer. En revanche, le temps d'exécution n'a lui été réduit que d'un facteur 3,5 au lieu de 4. Même si l'utilisation de 4 processeurs engendre bien une réduction du temps d'exécution de l'algorithme d'un facteur 4 en revanche, le taux de rupture de pipeline n'a lui pas été réduit d'un facteur 4. Ceci explique que le temps d'exécution global (Temps d'exécution + temps dû aux ruptures de pipeline) n'est pas réduit d'un facteur 4 comme attendu. La réduction du temps d'exécution étant inférieur à celle attendu va entraîner une augmentation de l'énergie globale consommée.
Il est à noter que dans ces résultats, nous n'avons pas pris en compte la consommation en mode de veille or, comme les processeurs travaillent moins longtemps que dans le cas monoprocesseur, il faut absolument prendre en compte la consommation du mode de veille. Le paragraphe suivant présentera donc les résultats obtenus avec la prise en compte de la consommation du mode de veille.

4.3 Résultats : cas multiprocesseur et mode de veille

Dans cette étude, nous considérerons que la contrainte de temps d'exécution est égale au temps d'exécution dans le cas monoprocesseur. Donc, une fois que les processeurs auront fini d'exécuter leur code, ils se mettront en veille profonde jusqu'à la prochaine itération de l'application. Le temps de veille sera alors égal au temps d'exécution dans le cas monoprocesseur moins le temps d'exécution dans le cas multiprocesseur. Dans le cas du TMS320C6201, la consommation de puissance en veille profonde est fonction de la fréquence d'horloge i.e. plus la fréquence est élevée et plus la consommation de puissance en veille est importante.
Pour éviter de mettre en veille les processeurs, nous pouvons diminuer la fréquence d'horloge dans le cas multiprocesseur afin de lisser la charge du processeur sur l'ensemble de la contrainte de temps (temps d'exécution en monoprocesseur). Cette diminution de la fréquence aura comme effet de diminuer la consommation de puissance des processeurs. La figure 4 présente les résultats pour différentes tailles d'image et différentes fréquences d'horloge.

Les résultats obtenus montrent que la solution multiprocesseur est toujours moins performante en terme de puissance et d'énergie que la solution monoprocesseur. En effet, même si le temps d'exécution est diminué par l'utilisation d'une architecture multiprocesseur, ce gain de temps est inférieur à la surconsommation de puissance engendrée par l'utilisation d'une telle architecture.
Nous voyons également que la consommation d'énergie ne diminue pas de façon linéaire lorsque la fréquence diminue comme c'est le cas en monoprocesseur. Ceci s'explique par le fait que la

consommation en veille n'est pas linéaire en fonction de la fréquence. Ces résultats montrent également que le point optimum en fréquence n'est pas celui ou la charge de calculs est lissée sur l'ensemble de l'espace temporel utilisable.
L'écart de performance en terme de consommation d'énergie se réduit dès lors que la taille d'image devient de plus en plus grande. Mais, même avec une taille de 1024x1024 pixels, la solution multiprocesseur consomme 1,09 fois plus d'énergie que la solution monoprocesseur.

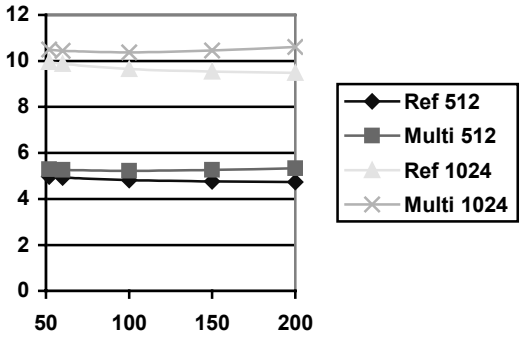

Figure 4. Résultats avec mode de veille

Que se passerait-il si on considérait maintenant non plus la seule fonction d'estimation de mouvement mais l'ensemble du codeur MPEG-2 ?
Considérons un processeur C6201 réalisant la totalité de l'application d'encodage MPEG-2 sur une image de taille 64x64 pixels et avec une fréquence d'horloge de 200MHz. Puis considérons un MPSoC intégrant 4 C6201 travaillant à 200MHz. Un des C6201 réalise la totalité de l'encodage mais seulement un quart de l'estimation de mouvement. Les 3 autres processeurs s'occupent de réaliser les trois autres quarts de l'estimation de mouvements.
Les résultats de cette étude sont présentés dans le tableau 3.

Texe (ms)	P (mW)	Pmax (mW)	E (mJ)
1 seul C6201			
1.068	3925	7385	4.192
MPSoC			
0.976	18311	29540	4.305

Tableau 3. Résultats sur l'encodeur MPEG-2

Le gain réalisé par l'utilisation du MPSoC en terme de temps d'exécution est de 8,6%, la surconsommation de puissance est de +366% et la surconsommation d'énergie est de 2,7%.
En conclusion, l'utilisation d'architectures de même type pour une solution multiprocesseur afin de remplacer une solution monoprocesseur est donc à déconseiller car elle n'apporte pas de gain en terme de consommation d'énergie.

Pour améliorer ces résultats la solution serait-elle d'utiliser une architecture MPSoC composée de processeur ayant des architectures « plus simples » ? Pour répondre à cette question, nous avons étudié la consommation d'un MPSoC utilisant non plus 4 C6201 mais 4 C5510 (DSP low power de TI). L'application est toujours l'encodeur MPEG-2 travaillant sur une image de taille 64x64 pixels. Les résultats de référence seront ceux obtenus avec une architecture monoprocesseur à base d'un C6201. Afin de respecter la contrainte de temps, il faudrait que le MPSoC puisse fonctionner à 310MHz. Supposons que tel est le cas ; les résultats de cette étude sont présentés dans le tableau 4.

Texe (ms)	P (mW)	Pmax (mW)	E (mJ)
1 seul C6201			
1.068	3925	7385	4.192
MPSoC (4 C5510)			
1.068	3003	3372	0.94

Tableau 4. Résultats sur l'encodeur MPEG-2

Avec une fréquence d'horloge de 310MHz, le temps d'exécution avec le MPSoC est identique à celui de la solution monoprocesseur. En revanche, le gain en puissance est de 23,5% et le gain en énergie est de 77,5%.
En conclusion, nous pouvons affirmer que l'utilisation d'architectures simples dans une plate-forme MPSoC est à privilégier ; Ces résultats concordent avec ceux présentés dans [1]. Il faut toutefois mettre en exergue que lors de ces études, nous n'avons pas pris en compte la consommation engendrée par la communication inter processeurs dans le MPSoC. Ces communications non pas forcément un impact négligeable en terme de consommation sur la consommation globale du MPSoC.

5. Conclusion et perspectives

Nos premiers résultats montrent que le meilleur compromis est obtenu pour l'utilisation d'une plate-forme utilisant des architectures simples. Ces résultats même s'ils confirment ceux exposés dans [1] sont toutefois à pondérer par le fait que nous n'avons pas tenu compte de la consommation engendrée par les communications.
Les travaux futurs devront répondre au problème poser par la modélisation de la consommation des communications dans les MPSoC. De plus, nous devrons déterminer quelle est l'architecture minimale à utiliser dans un MPSoC afin d'optimiser au mieux la consommation d'énergie d'une application.

Bibliographie

[1] H. De Man " Connecting E-Dreams to Deep-Submicron Realities ", *IEEE PATMOS 04*, Springer Verlag LNCS vol. 3254, pp. 1, September 2004.

[2] H. Baaker, "Powering up the mobile phone," Speech Technology Journal, Nov./Dec. 2001.

[3] N. Julien, J. Laurent, E. Senn, E. Martin, "Power consumption modeling of the TI C6201 and characterization of its architectural complexity", *IEEE Micro*, Special Issue on Power- and Complexity-Aware Design, Sept./Oct. 2003.

[4] E. Senn, J. Laurent, N. Julien, E. Martin "SoftExplorer: Estimation, Characterization, and Optimization of the Power and Energy Consumption at the Algorithmic Level", *IEEE PATMOS 04,* Springer Verlag LNCS vol. 3254, pp. 342-351, September 2004