Techniques de réduction de la consommation dans les systèmes embarqués temps-réel

Frédéric Parain, Michel Banâtre, Gilbert Cabillic, Teresa Higuera, Valérie Issarny, Jean-Philippe Lesot

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Techniques de réduction de la consommation dans les systèmes embarqués temps-réel

Frédéric Parain — Michel Banâtre — Gilbert Cabillic — Teresa Higuera — Valérie Issarny — Jean-Philippe Lesot

N° 3932

Mai 2000

THÈME 1

ISSN 0249-6399 ISRN INRIA/RR--3932--FR

INRIA
RENNES

Techniques de réduction de la consommation dans les systèmes embarqués temps-réel

Frédéric Parain* , Michel Banâtre* , Gilbert Cabillic* , Teresa Higuera† , Valérie Issarny† , Jean-Philippe Lesot*

Thème 1 — Réseaux et systèmes
Projet Solidor

Rapport de recherche n° 3932 — Mai 2000 — 30 pages

Résumé : L'évolution actuelle des systèmes embarqués tend à leur faire intégrer une puissance de traitement de plus en plus importante tout en conservant, voir en améliorant, leur autonomie énergétique. Si depuis longtemps des techniques de diminution de la consommation ont été recherchées, elles deviennent maintenant primordiales dans l'élaboration d'un système embarqué.

Ce document présente un certain nombre de techniques permettant de réduire la consommation énergétique. Certaines solutions sont purement matérielles, d'autres purement logicielles et d'autres enfin, appelées hybrides, nécessitent une collaboration entre une partie matérielle et une partie logicielle. Une de ces solutions hybrides est étudiée plus en détails : l'adaptation dynamique de la tension d'alimentation. Cette technique illustre l'impact que peut avoir une technique de diminution de la consommation sur un système ; dans ce cas précis, la modification de l'ordonnanceur d'un système temps-réel.

Mots-clés : consommation, système embarqué, ordonnancement temps-réel, adaptation dynamique de la tension

* {Prénom.Nom}@irisa.fr
† {Prénom.Nom}@inria.fr

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 Rennes Cedex (France)
Téléphone : +33 2 99 84 71 00 —Télécopie : +33 2 99 84 71 71

Power reduction in real-time embedded systems

Abstract: Embedded systems are evolving to offer higher computation capacity. Nevertheless they also have to offer a large energetic autonomy. Power conservation technics have been developped for a long time but they are now becoming one of the main concern of the design of an embedded system.

This document describes several technics developped to decrease power consumption in embedded systems. This includes hardware solutions as well as software or co-design solutions. One of this co-design solution, called dynamic voltage scaling, is studied more precisely. It illustrates the impact a power conservation technic can have on a system, in particular on scheduling in real-time systems.

Key-words: power, embedded systems, real-time scheduling, dynamic voltage scaling

Table des matières

Introduction

L'essor de l'informatique mobile allié à un besoin sans cesse croissant en performances concourent à l'avènement d'un nouveau type de systèmes embarqués. La principale caractéristique de ces nouveaux systèmes est la combinaison d'une puissance de calcul importante avec une grande autonomie de fonctionnement, or ces deux paramètres sont antagonistes. En effet, les technologies actuelles ne permettent l'augmentation de la capacité de traitement des processeurs qu'au prix d'une augmentation considérable de leur puissance. De nombreuses techniques, matérielles et logicielles, ont été développées afin de maîtriser cette consommation. Malgré cela, la gestion de la consommation revêt une importance de plus en plus grande dans la conception d'un système embarqué, et affecte de plus en plus de composants au sein d'un tel ensemble. Les derniers travaux menés sur l'adaptation dynamique de la tension vont même jusqu'à modifier l'ordonnanceur du système d'exploitation ; modifications qui peuvent avoir de graves conséquences dans le cas des systèmes temps-réels où l'ordonnanceur est responsable du comportement temporel des applications.

La première partie est un bref rappel sur les systèmes embarqués. La seconde partie est consacrée aux diverses techniques, tant matérielles que logicielles, qui permettent d'agir sur la consommation. L'une de ces techniques, l'adaptation dynamique de la tension d'alimentation, est étudiée plus en détail dans la troisième et dernière partie.

1 Systèmes embarqués

1.1 Introduction

Le moyen le plus simple de définir un système embarqué pourrait être : « système qui n'est pas relié physiquement à dispositif fixe », mais si cette définition a le mérite d'être simple, elle ne permet pas d'en connaître directement les caractéristiques. C'est pour cela qu'il est plus judicieux de définir un système embarqué comme une unité de traitement possédant une certaine autonomie. Cette autonomie est la propriété la plus couramment associée à la notion de système embarqué et elle peut se décliner sous plusieurs formes.

1.1.1 Autonomie de fonctionnement

Afin de pouvoir fonctionner de manière autonome, le système embarqué doit disposer de l'ensemble minimum des éléments physiques d'un système: processeur, mémoire, entrées/sorties, ainsi que d'un programme contrôlant ces différents éléments. Il doit également disposer d'une source d'énergie : générateur ou batteries, la nature de cette source pouvant influer sur le comportement du système, avec, par exemple, une gestion de la consommation dans le cas des batteries.

1.1.2 Autonomie fonctionnelle

La différence entre autonomie de fonctionnement et autonomie fonctionnelle apparaît au niveau des services. Un système embarqué offrant une autonomie fonctionnelle est capable de fournir ses services sans aucun recours à des composants qui lui sont externes, alors qu'un système embarqué sans autonomie fonctionnelle dépend toujours de structures extérieures. Un téléphone portable, par exemple, peut fonctionner parfaitement (système complet et batteries pleines), mais ne pas être capable de fournir le service de communication attendu s'il se trouve hors zone de réception des cellules GSM. Il s'agit donc d'un système autonome du point de vue du fonctionnement mais pas du point de vue des fonctionnalités. En revanche, une calculatrice de poche présente une autonomie fonctionnelle évidente.

Un système peut également présenter une certaine autonomie fonctionnelle, mais offrir des services plus étendus lorsqu'il se trouve dans un environnement adéquate. Un cas typique est celui de l'ordinateur portable équipé d'un modem, il possède une autonomie fonctionnelle, mais on peut le relier à l'Internet ou à un serveur distant à l'aide d'un réseau filaire ou cellulaire afin de profiter de nouveaux services offerts par des prestataires distants.

1.2 temps-réel

Beaucoup de systèmes embarqués interagissent directement avec leur environnement, afin d'obtenir des valeurs d'entrées (capteurs) ou bien pour appliquer le résultat d'un traitement (actionneur). Mais certains ont des contraintes temporelles nettement plus fortes que d'autres. Qu'un PDA[1] mette une ou trois secondes à afficher l'adresse que l'on recherche dans son annuaire, cela n'est pas un problème quand l'information est toujours utilisable malgré les deux secondes de retard. Mais si un téléphone portable a plusieurs dizaines de millisecondes de retard dans le décodage des trames GSM, la conversation deviendra incompréhensible pour l'utilisateur. Les systèmes embarqués peuvent donc être confrontés à des environnements logiciels extrêmement hétérogènes pouvant aussi bien inclure des contraintes temporelles fortes qu'une absence totale de notion temporelle.

Deux techniques sont utilisées pour satisfaire les contraintes temporelles : les développements monolithiques et les RTOS[2]. Les développements monolithiques sont effectués dans un langage de bas niveau, ils consistent à écrire un seul programme qui aura la charge du fonctionnement du système dans son intégralité. Le respect des contraintes temporelles est obtenu par la conception même du programme. À l'inverse, les RTOS fournissent aux développeurs des mécanismes adaptés à la programmation temps-réel comme des ordonnanceurs à priorités, l'héritage des priorités [SRL90] lors des blocages sur des ressources partagées, et des mécanismes de communications en temps borné. Ces RTOS permettent l'utilisation de techniques plus évoluées, comme l'analyse *RMA*[3] [LL73] pour le développement des applications temps-réels.

[1] *Personnal Digital Assistant* ou agenda électronique de poche

[2] *Real-Time Operating System* ou Système d'Exploitation Temps-Réel

[3] *Rate Monotonic Analysis*, algorithme d'ordonnancement temps-réel attribuant aux tâches des priorités proportionnelles à leur fréquence

1.3 Disponibilité

La disponibilité caractérise la capacité d'un système à assurer à tout moment un service donné. Elle est souvent associée au caractère critique que peuvent avoir les conséquences de l'échec du service. Un téléphone portable défaillant ne tend pas à provoquer une catastrophe, alors qu'un stimulateur cardiaque qui s'arrête met immédiatement en danger la vie de son porteur. Mais un appareil peut supporter plusieurs applications, ou plusieurs modes de fonctionnement plus ou moins critiques : ainsi une communication téléphonique peut devenir critique si elle est destinée à un service d'urgence. D'autres systèmes comme les commandes de vol d'avion combinent un fort caractère critique avec des contraintes temporelles strictes. Cet aspect des systèmes embarqués concerne essentiellement les travaux effectué sur la sûreté de fonctionnement.

2 Consommation

Un problème souvent rencontré par les systèmes embarqués, notamment ceux considérés dans ce document, est celui de l'autonomie. Certains systèmes comme le système de freinage ABS[4] qui équipe certains véhicules, n'ont pas ce genre de problèmes, car leur alimentation provient de la génératrice du moteur, et si jamais celui-ci s'arrête faute de carburant, le système ABS n'a plus de raison de fonctionner puisque le véhicule est arrêté. Mais, une grande majorité des systèmes embarqués (téléphones cellulaires, ordinateurs portables) sont confrontés à ce problème d'autonomie.

Pour étendre l'autonomie de fonctionnement d'un système, seules deux méthodes existent : augmenter la quantité d'énergie embarquée ou diminuer la consommation du système. La première solution a entraîné de nombreuses recherches dans le domaine des batteries, mais malgré les progrès effectués dans ce domaine, il est toujours difficile d'augmenter la capacité d'une batterie sans en augmenter le poids, le volume et le prix. La seconde solution est complémentaire à la première, et a également donné lieu à de nombreuses recherches dans le domaine de l'électronique et de l'informatique.

Cette partie porte sur les techniques utilisées pour concevoir un système embarqué à faible consommation, ces techniques sont regroupées en trois catégories : les techniques matérielles, les techniques logicielles et enfin les techniques de conception hybrides requièrant une collaboration entre le matériel et le logiciel.

Il existe plusieurs méthodes pour obtenir un système à faible consommation. La première et la plus répandue (cf 2.1), est la conception de composants spécifiquement conçus pour consommer le minimum d'énergie. La seconde méthode (cf 2.2) consiste à modifier le logiciel pour diminuer le coût énergétique de son exécution : optimisation du code des applications, adaptation des protocoles de communication et exécution déportée. Enfin la troisième méthode (cf 2.3) consiste à réaliser une collaboration entre le logiciel et le matériel afin d'optimiser la consommation totale de l'appareil. Elle s'appuie sur des mécanismes matériels pour diminuer la consommation mais utilise le logiciel pour réaliser une adaptation

[4] *Anti-lock Braking System* ou Système de freinage anti-blockage

dynamique de la consommation en fonction de la charge courante du système. Cette méthode est décrite plus précisément dans le cadre de la mise en veille des périphériques ainsi que sur l'adaptation dynamique de la vitesse du processeur.

2.1 Techniques matérielles

2.1.1 Technologie des composants

La technologie des composants électroniques est un domaine où les progrès se succèdent à un rythme extrêmement rapide. L'augmentation des fréquences de fonctionnement des processeurs, ainsi que la réduction de la taille des circuits ($1.5\mu m$ pour le processeur Intel 80386, $0.8\mu m$ pour le Pentium, $0.25\mu m$ pour le Pentium II) permettent une intégration toujours plus fine, concentrant une puissance de plus en plus importante dans un espace toujours plus étroit. Mais, en ce qui concerne les processeurs généraux, l'amélioration des performances s'est longtemps effectuée sans aucune préoccupation de la consommation [TSR+98]. Ce n'est qu'avec l'explosion du marché des systèmes embarqués, ainsi que l'apparition des problèmes de surchauffe des composants, que cet aspect a été pris en compte au niveau de la conception. Les trois techniques décrites ci-dessous, c'est-à-dire la diminution de la tension d'alimentation, les activations séparées de blocs logiques et le contrôle du taux de basculement des bits[5], montrent les différentes solutions développées pour limiter la consommation des composants.

Tension d'alimentation La puissance dissipée par un composant est proportionnelle à sa fréquence de fonctionnement (f), à sa capacité équivalente (C) et au carré de sa tension d'alimentation (V) [RS98], ce qui est souvent exprimé par la formule suivante:

$$P = \alpha.C.f.V^2 \quad (1)$$

La tension d'alimentation est donc, grâce à son élévation au carré, le facteur dont la réduction aura l'impact le plus important sur la consommation. Au cours de ces dernières années, les tensions d'alimentation n'ont pas cessé de diminuer, initialement de 5V, c'est maintenant la valeur 3.3V qui est la plus répandue, et certains systèmes peuvent descendre jusqu'à 1.1V [BBB+95]. Cette diminution n'est pas seulement due aux progrès dans la conception des processeurs, mais aussi à une meilleure maîtrise des techniques de fabrication. En effet, une tension plus basse nécessite le recours à une technologie de plus grande finesse ($0.8\mu m$ pour une tension de 5.5V, $0.5\mu m$ ou $0.35\mu m$ pour une tension de 3.3V) [TSR+98]. Cependant ces progrès se heurtent de plus en plus à des problèmes liés aux lois de la physique. L'abaissement des tensions d'alimentation, et donc des tensions de seuil, a des répercutions au niveau de l'effet transistor, et les courants de fuite qui apparaissent auront un impact de plus en plus important sur la consommation et le fonctionnement du circuit. Dans le même sens, la diminution de la tension diminue le rapport signal/bruit, et rend le composant plus sensible aux perturbations externes [CGX96].

[5] appelé *bit-switching* dans la terminologie anglo-saxonne, cela correspond au basculement d'un bit d'un état à un autre, de 0 à 1 ou de 1 à 0

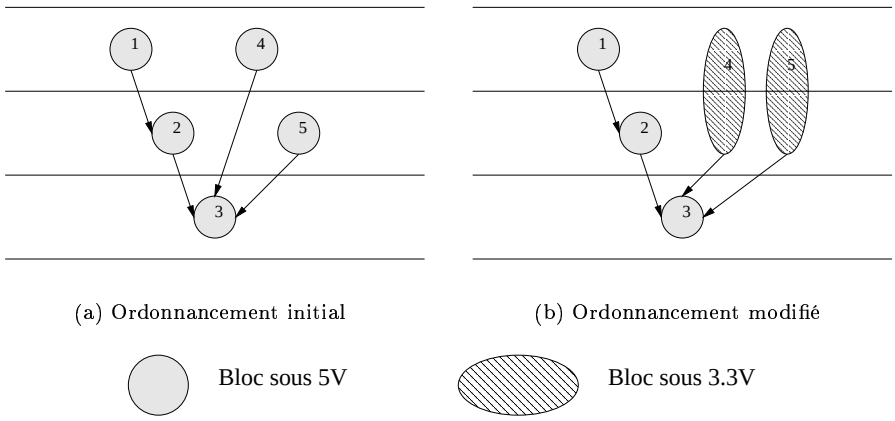

FIG. 1: Graphes des Flots de Données (DFG)

Activations séparées Une autre méthode pour diminuer la consommation d'un composant est de limiter son alimentation aux blocs nécessaires au traitement en cours. Cette technique a été utilisée dans [SD95] pour diminuer la consommation des caches sous le nom de *sub-banking*. Elle consiste à diviser une mémoire cache en plusieurs blocs (*bank*), pouvant être activés indépendamment les uns des autres. Lors d'un accès mémoire à travers le cache, le numéro de bloc est immédiatement extrait à partir de l'adresse demandée, et seul le bloc mémoire correspondant est activé (généralement cela consiste à élever la tension des pistes du bus de données à la moitié de la tension de seuil afin d'augmenter la rapidité du cache). Si la mémoire cache est divisée en N blocs, l'énergie consommée à chaque activation sera égale à $1/N$ fois l'énergie nécessaire à un cache sans *sub-banking*. Cela ne dégrade pas les performances du cache puisque que le décodage du bloc à activer peut être effectué en parallèle de la vérification des *tags*[6] du cache, mais nécessite l'implantation d'un circuit de décodage supplémentaire. Cette technique ne peut cependant être appliquée que sur la partie du cache qui contient les données de la mémoire, celle qui contient les *tags* ne peut pas être optimisée de cette manière car il n'est pas possible de prédire quelle zone des *tags* sera utilisée.

Une autre méthode consiste à utiliser plusieurs tensions d'alimentation en fonction des besoins des différents blocs. Les processeurs actuels sont conçus à l'aide de bibliothèques permettant de décrire un composant par son comportement au niveau RTL[7]. Cette description peut être mise sous la forme d'un graphe appelé DFG[8] dont une représentation est donnée dans la figure 1.

Dans ces graphes DFG, les nœuds représentent des blocs de base (additionneur, multiplexeur, ...), et les arcs indiquent les dépendances, c'est-à-dire le cheminement des don-

[6]Indicateurs sur l'identité et la validité des données présente dans le cache
[7]Register-Level-Transfer
[8]Data Flow Graph, ou Graphe des Flots de Données

nées d'un bloc à l'autre. Les lignes horizontales servent à délimiter les différents cycles de l'exécution. Comme cela a été expliqué précédemment, une diminution de la fréquence de fonctionnement d'un composant permet de diminuer sa tension d'alimentation, et donc sa consommation. L'idée développée dans [LHW97] est de déterminer pour chaque bloc quelle peut être sa vitesse minimale compte-tenu de ses dépendances et des blocs qui dépendent de son résultat. Et si la vitesse d'un bloc peut être diminuée, on diminue alors également sa tension d'alimentation. La figure 1(a) montre un ordonnancement possible avec des blocs alimentés à 5V. L'exécution des blocs 4 et 5 peut être modifiée de manière à obtenir l'ordonnancement représenté par la figure 1(b), ce qui permet de remplacer ces blocs 4 et 5 initiaux par des blocs assurant la même fonction mais plus lentement et ne requièrant que 3.3V comme tension d'alimentation.

Afin de déterminer quels sont les blocs susceptibles d'être remplacés, les algorithmes ASAP (*As Soon As Possible* qui place les blocs au plus tôt, c'est-à-dire dès que toutes leurs valeurs d'entrées sont disponibles) et ALAP (*As Late As Possible* qui place les blocs au plus tard, c'est-à-dire juste avant que d'autres blocs ne se retrouvent bloqués en attente de leurs résultats) sont appliqués afin de connaître la durée sur laquelle peut-être réparti le traitement de chaque bloc. Le remplacement d'un bloc s'effectue donc en fonction du temps disponible et des technologies existantes. En effet, seuls des blocs fonctionnant à des tensions standards (5V, 3.3V, 2.9V, 2.45V, 1.6V, 1.1V, ...) sont utilisés.

Ces techniques permettent donc d'ajuster au mieux la consommation totale d'un circuit, mais elle se limite aux langages de description de circuits. Les répercutions sur la réalisation proprement dite ne sont pas abordées, notamment la cohabitation de circuits ayant différentes tensions d'alimentation, donc différentes tensions de signaux, et partageant les mêmes bus.

Basculement de bit Dans l'équation (1) de la puissance donnée précédemment le paramètre α masque plusieurs phénomènes complexes du fonctionnement d'un processeur qui ont un effet sur la consommation totale, l'un d'eux est le basculement de bit[9]. Chaque changement d'état (passage de l'état "1" à l'état "0" ou inversement) entraîne une consommation due à la capacité équivalente de la piste qui subit ce changement. C'est pour cela que les travaux décrits dans [MC95] cherchent à diminuer le taux de basculement des bits en optimisant la réutilisation des informations présentent sur les bus. Cette optimisation se situe au même niveau que les activations séparées présentées précédemment, mais cette fois ci l'objectif est de regrouper les blocs susceptibles d'être activés sur les mêmes données autour d'un même bus. Ainsi, une fois ces données présentent sur le bus, elles pourront être utilisées par plusieurs blocs.

Une autre méthode pour réduire le nombre de changements d'états des bits est d'utiliser le codage de Gray pour les adressages mémoires plutôt que la traditionnelle représentation en complément à deux. La particularité de ce codage est de ne présenter qu'un seul bit de différence entre un nombre et son successeur, comme on peut le voir sur la figure 2.

[9] *bit-switching* dans la terminologie anglo-saxonne

Valeur décimale	Code en complément à deux	Code de Gray
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100

FIG. 2: Code de Gray

L'utilisation d'un tel code est donc très appropriée pour le codage des adresses mémoires dont les accès sont souvent séquentiels. Il faut cependant être prudent dans l'évaluation de l'influence d'une telle méthode dans un environnement réel, car la mise en œuvre est loin d'être aisée. Tout d'abord les composants standards (additionneurs, décodeurs, ...) ont été conçus pour fonctionner avec un codage en complément à deux, et ne peuvent donc pas être utilisés avec le code de Gray. Cela implique donc de concevoir un nouveau jeu de composants, spécifiquement étudiés pour le codage de Gray. De plus, si la séquentialité des accès mémoire est relativement bien suivie, cet effet est très vite masqué par les différents étages de mémoire cache. Ainsi plus la requête remonte dans la hiérarchie de la mémoire, et donc plus les accès aux bus sont coûteux, moins la séquentialité est respectée, et donc moins l'utilisation du codage de Gray est justifiée. C'est sans doute à cause de ces problèmes d'implantation que dans [SD95] aucune mesure de performance n'est donnée pour cette méthode.

2.1.2 Architecture du matériel

L'optimisation d'un système ne s'effectue pas seulement au niveau de la conception de ces composants, mais également au niveau du choix d'une architecture générale. Les deux exemples ci-dessous illustrent l'influence que peuvent avoir des décisions de haut niveau sur la consommation, le premier porte sur le choix d'une technologie par rapport à une autre, et le second concerne le problème du dimensionnement appliqué aux caches.

Choix des périphériques Le choix d'un périphérique peut devenir un problème critique lorsque celui-ci est destiné à un système dont la gestion de la consommation représente un critère essentiel. Mais au lieu de rechercher le périphérique le plus performant de sa catégorie, il peut parfois être préférable de changer complètement de technologie. Un exemple présenté dans [RS98] concerne le périphérique de mémoire de masse (ou mémoire permanente). Le périphérique le plus couramment utilisé est le disque dur, mais il s'agit d'un périphérique extrêmement gourmand en énergie (notamment à cause de la présence de systèmes mécaniques

et non purement électroniques). Le remplacer par un système électronique est désormais possible grâce au développement des mémoires flash. Ces mémoires qui conservent leurs données même lorsqu'elles ne sont plus alimentées ont une consommation nettement plus faible, ce qui permet de diminuer de 60 à 90% l'énergie nécessaire au stockage des données [RS98]. Mais les caractéristiques des mémoires flash étant très différentes de celles des disques durs, certain mécanismes systèmes doivent être adaptés en conséquence. Les temps d'accès en lecture des mémoires flash étant du même ordre de grandeur que ceux d'une mémoire dynamique, il n'est plus nécessaire d'utiliser des caches systèmes en lecture. Cependant les temps d'écriture étant plus importants, les caches en écriture peuvent être conservés. L'effacement des données sur une mémoire flash est problématique, en effet, les données ne peuvent être détruites que par bloc, ce qui rend remet en cause l'implantation de nombreux systèmes de fichiers. Mais les systèmes à base de journaux (*log-structured file system*) ne sont pas perturbés par cette contrainte car la mise à jour des informations se fait par l'ajout d'une entrée dans le journal. Cependant les mémoires flash ont un nombre de cycles écriture/effacement assez limité par rapport à d'autres périphériques (de l'ordre de 100000 cycles pour chaque bloc de données). Le système de fichier doit donc veiller à répartir les cycles d'utilisation sur la totalité des blocs pour éviter de voir s'user prématurément les blocs les plus sollicités [RS98].

Dimensionnement Le dimensionnement des périphériques a également une influence sur la consommation. Un des exemples les plus flagrants est le dimensionnement des caches. Plus les caches ont une taille importante, plus ils consomment. Mais plus un cache est grand, plus sont taux d'échec[10] est faible, et donc moins il génère d'accès vers le reste de la hiérarchie mémoire. Ces accès étant plus coûteux en temps et en énergie qu'un accès aux données présentent dans le cache, un cache de grande taille aura donc également tendance à diminuer la consommation globale de la mémoire.

Plusieurs études ont montré que du point de vue de la consommation, il était plus intéressant d'avoir plusieurs niveaux de caches plutôt qu'un seul niveau de cache de grande taille [KBN95] [HWO97]. Mais aucune méthodologie n'est proposée pour déterminer le nombre, le type et la taille de ces caches, car il s'agit toujours de compromis à établir entre les caractéristiques de la plate-forme matérielle et les besoins de l'application. Les études relatives à ce problème de dimensionnement utilisent des *benchmarks*[11] pour évaluer les différentes configurations [SD95] [HWO97]. Les courbes de la consommation en fonction de la taille du cache présentent généralement un minimum [HWO97], ce qui facilite le dimensionnement, mais il est généralement nécessaire de comparer plusieurs configurations d'associativité afin de trouver la plus économe [SD95]. Ce dimensionnement dépend également des applications qui sont exécutées (régularité du code, des données, taux de réutilisation), cela ne pose pas de problèmes dans le cas d'un environnement statique parfaitement maîtrisé et déterminé, mais dans le cas des environnements dynamiques il n'est pas possible d'effectuer ce type

[10] *miss-rate* dans la terminologie anglo-saxonne
[11] Séries de tests d'évaluation

de dimensionnement. Il est cependant envisageable d'utiliser un échantillon représentatif d'applications types pour effectuer l'évaluation.

2.1.3 Circuits reconfigurables

Au lieu de rechercher le dimensionnement le plus approprié pour les différents composants, il peut être envisager de les spécialiser en fonction de l'usage qui en est fait. La technologie des FPGA[12] se prête très bien à ce type d'optimisation. L'architecture proposée dans [AR96] cherche à exploiter ce concept : un processeur central générique et des processeurs satellites spécifiques, programmés en fonction des applications visées. Les reconfigurations des processeurs satellites sont effectuées pour des traitements très spécifiques et parfaitement caractérisés, comme le codage de la voix ou le décodage de trames CDMA[13] [Rab97]. Les gains sur la consommation ainsi obtenus sont remarquables [ASWR98], dépassant même ceux obtenus sur les DSP faible consommation grâce à une meilleure optimisation en fonction du traitement effectué. Le prix à payer pour cette spécialisation est que chaque processeur satellite ne peut plus effectuer qu'un seul type de traitement, il faut donc autant de processeurs satellites qu'il existe de traitements dont on veut optimiser la consommation, et ces processeurs spécialisés ne seront plus utilisables pour des traitements autre que le leur.

2.2 Techniques logicielles

Elles consistent à modifier le code à exécuter afin de diminuer la consommation induite par l'exécution de ce dernier. Elles interviennent au niveau des outils de développement des applications, mais bien que leurs concepts restent relativement simples, il est difficile d'automatiser leur utilisation. En effet elles requièrent une connaissance très précise du code applicatif ainsi que des caractéristiques de la plate-forme matérielle. C'est pourquoi nous étudierons les moyens existant pour quantifier la consommation de l'exécution d'un programme avant de décrire quelques optimisations possibles.

2.2.1 Évaluation de la consommation logicielle

Malgré les études menées dans [TMW94] il est très difficile d'évaluer le coût énergétique d'une instruction sur les processeurs actuels. L'utilisation systématique de *pipeline* ne permet pas d'isoler la consommation d'une instruction parmi toutes celles présentent dans le *pipeline*. Il est cependant possible d'obtenir une valeur moyenne du coût énergétique d'une instruction en exécutant celles-ci un grand nombre de fois, afin que le *pipeline* ne contienne que des instances de l'instruction que l'on chercher à évaluer. Cette méthode a cependant l'inconvénient de ne pas pouvoir prendre en compte les interactions qui peuvent exister entre l'exécution de deux instructions différentes. Il a d'ailleurs été mesuré [TMW94] que la consommation de deux instructions exécutées successivement était toujours supérieure à la

[12] *Field-Programmable Gate Array*, circuit électronique programmable électriquement
[13] *Code Division Multiple Access*

somme de leurs consommations respectives. Cette différence, appelée *circuit state overhead* est fonction des deux instructions exécutées. Si la participation de ce surcoût à la consommation globale est faible sur les processeurs étudiés (la famille x86 d'Intel), la question se pose de savoir si elle sera également faible pour d'autres types de processeurs (RISC ou DSP). Mais la valeur de la consommation mesurée pour chaque instruction ne représente qu'une valeur de base qui doit encore être modulée en fonction des valeurs d'entrée, car la consommation d'une simple instruction `MOV`[14] peut varier de 14% en fonction du nombre de bits à 1 de la valeur à transférer [TMW94]. Il existe également d'autres facteurs pouvant intervenir, comme les défauts de cache, mais cela dépasse le domaine des instructions car il faut alors prendre en compte l'environnement d'exécution des applications.

2.2.2 Optimisations de code

Bien que la consommation due à l'exécution d'un programme soit difficile à estimer, il existe une relation entre le nombre d'instructions exécutées et la consommation [LH98]. D'une manière générale, plus le nombre d'instructions est élevé, plus la consommation est importante, les techniques d'optimisation logicielles peuvent donc aider à diminuer cette consommation. La majorité des développements se font actuellement avec des langages dits de « haut-niveau », comme le langage C ou Ada. Les développements en langage de « bas-niveau » comme l'assembleur étant limités aux applications ayant des contraintes extrêmement fortes sur leurs performances et leurs fiabilités (programmation de DSP ou de codes critiques). L'avantage des langages de « haut-niveau » est de permettre au programmeur d'écrire son application dans un langage facilement compréhensible et de plus portable, la génération de code machine pour une architecture donnée étant réalisée de manière automatique par un compilateur. Cette méthode permet d'améliorer la vitesse de développement, mais cela s'effectue au dépend de la qualité des codes machines générés. Il est toujours possible d'optimiser « à la main » des codes ainsi générés, mais de tels codes ne sont généralement pas très lisibles leur manipulation est donc fastidieuse, et la taille importante des applications actuelles rend leur analyse encore plus complexe. Cependant, sur de petits programmes, il peut-être envisagé d'avoir recours à une telle méthode, comme c'est le cas dans [TMW94], où les opérations de mémoire à mémoire sont remplacées par des opérations de registre à registre, moins coûteuses en énergie. Cette optimisation permet de diminuer de 40% la consommation totale du programme, mais il s'agit d'un cas précis, et il n'est pas justifié de prendre cette valeur comme référence. Les optimisations de code manuelles étant difficile à mettre en œuvre, les techniques décrites ci-dessous portent sur des optimisations automatiques au niveau du générateur de code. Elles n'atteignent pas le même niveau de performances mais elles sont nettement plus simples et peuvent être aisément implantées dans une chaîne de développement.

Fonctions en ligne et déroulement de boucles Les techniques d'expansion des fonctions en ligne et de déroulement des boucles (dénommées respectivement *inlining* et *loop-*

[14] instruction assembleur recopiant une donnée d'un endroit de la mémoire vers un autre

unrolling dans la terminologie anglo-saxonne) sont bien connues des programmeurs qui cherchent à optimiser leurs programmes avec comme critère le temps d'exécution. L'utilisation de ces techniques peut, dans certains cas, diminuer la consommation d'un programme. L'appel de fonction est une opération longue et coûteuse dans de nombreux environnements : stockage des paramètres, création d'un nouveau contexte (*frame*[15]), exécution de la fonction, sauvegarde de la valeur de retour, destruction du contexte, et enfin destruction des paramètres. Certains matériels propose un support pour l'appel de fonction, comme les fenêtres de registres [16] des architectures SPARC[GW94], qui permettent d'optimiser l'utilisation des registres d'une fonction à une autre. Cela permet de diminuer le surcoût en temps que provoque la sauvegarde d'un banc de registres en mémoire centrale, et également diminue la consommation car les opérations restent au niveau du processeur et ne circulent pas sur les bus externes. Cependant, un nombre trop important d'appels de fonction imbriqués peut dépasser les capacités du matériel et obliger le système à sauvegarder, malgré tout, une partie des registres en mémoire centrale. Cette sauvegarde ne peut s'effectuer qu'à l'aide d'instructions de transfert de registres à mémoire, beaucoup plus coûteuses en énergie que les instructions de registres à registres utilisées pour travailler à l'intérieur d'une fenêtre de registres [LH98].

La technique des fonctions en ligne consiste à recopier le code d'une fonction à l'endroit où elle est invoquée plutôt que d'insérer une procédure d'appel de fonction. Le code de la fonction peut alors être exécuté sans qu'il y ait de surcharge due à un appel de fonction. Cette technique diminue également le nombre d'appels de fonction imbriqués, et donc le risque de dépassement des capacités du matériel et réduit donc le risque d'une sauvegarde des registres. La diminution du nombre d'instructions exécutées ainsi qu'une plus grande probabilité de travailler avec des instructions sur les registres plutôt qu'avec des instructions sur les mémoire permettent d'obtenir un code moins gourmand en énergie. L'inconvénient majeur de cette méthode reste l'augmentation de la taille du code, en effet, les fonctions en ligne sont recopiées autant de fois dans le code qu'elles sont appelées. C'est pour cette raison que ce sont généralement des fonctions de petite taille que l'on cherche à mettre en ligne. Cet impact sur la taille peut avoir une influence sur la consommation, un code de plus grande taille nécessite une mémoire plus grande pour être stockée. Or les mémoires sont des composants extrêmement gourmands en énergie et leur consommation croit avec leur taille, il y a donc un compromis à trouver entre la taille du code et la diminution des appels de méthodes.

Le déroulement de boucles comme son nom l'indique, agit sur les boucles. Ces dernières sont constituées d'une partie de traitement et d'une partie de contrôle qui sert à modifier les compteurs de boucles et à tester les conditions de fin. Le déroulement consiste donc à recopier plusieurs fois la portion de code correspondant au traitement, ainsi la proportion de temps passée dans le code de contrôle par rapport au temps de traitement global de la boucle, diminue. Là encore on peut obtenir une diminution du nombre d'instructions exécutées, et donc une diminution de la consommation. Il y a cependant des inconvénients, notamment

[15] Nouvel environnement de pile contenant entre autres les variables locales
[16] *registers window* dans la terminologie anglo-saxonne

une augmentation de la taille du code, dont l'un des effets de bord peut être que le code dépasse la capacité du cache d'instructions, ce qui entraîne une augmentation des accès en mémoire centrale et donc une augmentation de la consommation. Cet phénomène peut également survenir lors de l'utilisation de l'expansion des fonctions en ligne. Le déroulement des boucles tend également à augmenter la taille globale du code de l'application et on retrouve la même situation qu'avec les fonctions en ligne : un compromis à trouver entre la taille du code et la taille de la mémoire.

Ces deux méthodes ne sont pas spécifiques au problème de la consommation énergétique des programmes, elles ont été développées dans le but de réduire le temps d'exécution des programmes et leur impact sur la consommation peut être considéré comme un effet de bord. Mais les options d'expansion de fonction en ligne et de déroulement de boucles sont disponibles dans de nombreux compilateurs, leur mise en œuvre est donc extrêmement simplifiée. Cependant, le dimensionnement de ces techniques requière une connaissance très poussée de l'architecture matérielle.

Instructions spécifiques Les techniques des fonctions en ligne et de déroulement des boucles présentées précédemment sont des techniques générales d'optimisation de code. Il est également possible d'optimiser un code selon l'architecture sur laquelle il va s'exécuter. En effet, certains processeurs offrent des instructions très particulières qui ne sont pas toujours utilisées par les compilateurs. Le cas étudié dans [LT95] concerne un DSP qui offre une instruction `LAB` chargeant deux registres en un seul cycle. L'utilisation de cette instruction en remplacement de deux instructions traditionnelles `MOV` (chargement d'un seul registre) permet une économie d'énergie de 34.50% à 50.55%, cette variation étant fonction des valeurs des données à transférer. La répercution sur la consommation globale des applications varie de 23.13% à 47.41% d'énergie économisée pour les filtres utilisés comme tests dans [LT95]. Mais l'utilisation de l'instruction `LAB` entraîne des contraintes supplémentaires dans le placement des données en mémoire. En effet, le DSP étudié possède deux mémoires et le chargement de deux registres en parallèles ne peut se faire que si les données à charger viennent chacune d'une mémoire différente. Un placement judicieux des données en mémoire est donc une condition indispensable à l'utilisation de l'instruction `LAB` et donc à la diminution de la consommation. Ce placement est obtenu à partir d'un graphe d'accès, où les nœuds représentent les différentes variables du programmes et où les arcs joignent les variables qui interviennent dans une même opération. L'optimisation du placement s'effectue en cherchant une partition du graphe maximisant le nombre d'arcs traversant la ligne de partitionnement. L'algorithme de partitionnement utilisé est basé sur le recuit simulé[17], la fonction d'énergie évalue pour chaque partitionnement la somme totale des coûts de transfert pour chaque opération du programme, compte-tenu du placement proposé.

Cette méthode d'optimisation montre qu'avec une seule instruction, des gains significatifs peuvent être obtenus au niveau de la consommation. Cependant une telle optimisation ne peut être envisagée que pour une architecture bien particulière, et ne garantit en aucun

[17] *simulated annealing* dans la terminologie anglo-saxonne : algorithme permettant de trouver l'extremum d'une fonction complexe

cas sa portabilité sur une autre plate-forme. Le gain au niveau de la consommation doit également être réévaluée pour chaque nouveau système.

2.2.3 Protocoles

Il peut être également intéressant de modifier un protocole de communication pour en diminuer la consommation, par exemple en regroupant des données à émettre afin de diminuer le nombre d'émissions/réceptions. Une adaptation du protocole *Wireless Ethernet* est proposée dans [KK98] afin de permettre la mise en veille du récepteur. Dans ce type de réseau les stations embarquées sont en permanence à l'écoute de la station GSM émettrice, de la même manière que sur un câble *Ethernet*, toutes les cartes sont à l'écoute des trames circulant sur le réseau, à la différence que cette écoute permanente consomme une quantité importante d'énergie. Or cette énergie est consommée inutilement si aucune des émissions n'est destinée au récepteur en question, il est donc intéressant de chercher à diminuer cette perte d'énergie en mettant le récepteur en veille lors des périodes de silence entre la station émettrice et la station embarquée. Le problème de cette mise en veille vient de l'émission possible de messages depuis la base[18] (station émettrice du réseau GSM) vers le poste embarqué qui peut avoir désactiver son module de réception. Le protocole proposé est le suivant : lorsque le poste GSM embarqué veut activer la veille de son récepteur, il commence par déterminer la durée maximale de la veille qu'il communique à la base. À partir de ce moment, il peut mettre son module de réception en veille, car la base GSM mettra dans une file d'attente toutes les trames qui lui sont destinées. Une fois la durée de la veille écoulée, la base GSM émet toutes les trames en attente vers le poste embarqué. De son côté, le poste GSM peut à tout moment interrompre la veille du module de communication pour reprendre la communication avec la base. Cette modification du protocole nécessite une modification de la base, afin qu'elle puisse gérer ces déconnexions et stocker les trames en attente d'émission.

Cette technique ne se limite donc pas à une modification du système embarqué mais affecte l'ensemble des infrastructures du réseau, elle n'est donc applicable que pour le développement de nouveaux réseaux de communication et ne peut pas être utilisée dans le cadre des réseaux existants.

2.2.4 Exécution distante

L'utilisation des réseaux sans fil de type *Wireless Ethernet* peut également permettre de diminuer la consommation d'un appareil embarqué en déportant certains de ses traitements sur un serveur distant [RRPK98]. Ainsi, le client embarqué envoie ses données vers un serveur, se met en veille en attendant que le serveur finisse de traiter ces données, puis récupère le résultat final émit par le serveur. Seulement les communications entre le client et le serveur ont également un coût énergétique, qui peut être élevé si les données à transmettre vers ou depuis le serveur sont volumineuses. Il est alors important de quantifier correctement, le coût d'une exécution locale, par rapport au coût d'une transmission des données vers un serveur distant afin de choisir la solution la moins onéreuse en énergie. L'évaluation du

[18] généralement appelée *baystation*

coût des communications avec le serveur reste cependant difficile à évaluer car elle dépend du trafic existant sur le réseau sans fil, si ce trafic est dense, le nombre de collisions risque d'être élevé, et les retransmissions alors engendrées peuvent augmenter considérablement le coût du transfert de données. L'évaluation de la consommation dû à une exécution locale est également difficile à évaluer, car l'énergie requise par une application peut varier fortement en fonction de son comportement, et nombre d'applications ont un comportement qui dépend de leur environnement ou des données qu'elles doivent traiter. Une solution à ce problème a été proposée dans [RRPK99], mais elle ne concerne que les applications exécutées régulièrement car pour choisir le mode d'exécution d'une application, local ou distant, elle se base sur des mesures effectuées lors des exécutions précédentes. Et il existe certains types d'applications, comme les applications interactives, qui ne sont pas du tout adaptées aux exécutions distantes.

2.3 Techniques hybrides

Les techniques hybrides, ou combinées, sont basées sur une collaboration entre les composants matériels et les composants logiciels constituant le système embarqué. Les mécanismes utilisés pour diminuer la consommation proviennent des capacités du matériel, mais les décisions d'activer ou non ces mécanismes sont prises par le logiciel car il peut effectuer des choix plus pertinents. La première technique de conception combinée étudiée concerne la mise en veille de périphériques, et la seconde l'adaptation dynamique de la vitesse du processeur.

2.3.1 Mise en veille

La mise en veille consiste à désactiver certains périphériques ou certaines parties de périphériques lorsque le système n'en a pas l'usage pendant un certain temps. Les périphériques se contentent généralement d'implanter des mécanismes pour supporter plusieurs états de fonctionnement (de l'activité totale à la mise en sommeil complète, voir leur déconnexion du système). Ces différents états de veille correspondent à des niveaux de consommation différents, fonctions de l'activation ou de la désactivation des sous-systèmes du périphérique. Plus l'état de veille est profond, plus la consommation est diminuée mais plus le temps de réveil sera long. La cause de ce retard peut être de deux natures : physique ou logique. Le retard physique est dû à une contrainte généralement d'ordre mécanique, c'est typiquement le cas avec les disques durs. La mise en veille d'un disque dur revient souvent à arrêter le moteur qui entraîne les disques, grand consommateur d'énergie. Lors du réveil du disque dur, il faudra remettre les disques en rotation, ce qui demandera un certain temps à cause de leur inertie [RS98]. Le retard logique, quand à lui, est plus lié à la mise en sommeil des circuits électroniques, notamment les mémoires. Lorsque l'on coupe l'alimentation électrique d'une mémoire dynamique, cette dernière perd son contenu, or si elle contenait le contexte de fonctionnement du périphérique, ce dernier doit être restauré pour que le périphérique soit de nouveau opérationnel. La mise en sommeil doit donc être précédée d'une sauvegarde du contexte par le système d'exploitation. Le redémarrage doit alors inclure le temps

de recherche et de restauration de ce contexte avant que le périphérique soit de nouveaux pleinement opérationnel.

Bien que les mécanismes de mise en veille soient implantés au niveau matériel, la prise de décision sur l'endormissement de telle ou telle partie du système s'effectue généralement au niveau du système d'exploitation. En effet, ce dernier est le seul à avoir une connaissance de l'ensemble des activités du système [RS98]. Les applications sont les plus à même de prédire leurs besoins futurs, mais elles n'ont qu'une connaissance limitée de l'architecture sur laquelle elles s'exécutent, celle-ci étant masquée par le système d'exploitation. Cette méconnaissance leur rend impossible l'implantation d'une politique d'endormissement adaptée à leurs comportements. Dans le cas de l'anticipation des accès aux fichiers, l'application peut avoir une connaissance exacte de ses besoins à venir, mais ne connaissant pas les périphériques stockant ces fichiers, elles ne sera pas capable de déterminer quels sont les périphériques pouvant être mis en veille. De plus chaque application s'exécute de manière indépendante, c'est-à-dire sans tenir compte des autres applications pouvant cohabiter sur le même système, or certaines décisions qui sembleraient bonne du point de vue d'une application pourrait être désastreuses pour une application concurrente. Quant aux périphériques eux-mêmes, ils ne perçoivent que des accès sporadiques, et n'ont donc pas une vision de l'activité réelle du système (qui peut être masquée notamment par des caches du système d'exploitation).

Le fait que le système d'exploitation soit le mieux placé pour décider des mises en sommeil ne signifie pas qu'il dispose de toutes les informations nécessaires. Les applications peuvent avoir des comportements irréguliers, et l'utilisateur lui-même est extrêmement difficile à modéliser. L'utilisateur est pourtant un critère prépondérant. En effet, comme cela a été décrit précédemment, toute mise en veille provoque une augmentation de la latence du périphérique concerné lors de son réveil, ce qui du point de vue de l'utilisateur correspond à une diminution des performances. Une politique d'économie d'énergie « agressive », visant à une économie d'énergie maximale peut entraîner une dégradation des performances si importante que l'utilisateur la trouvera inacceptable et désactivera le mécanisme de gestion de la consommation. Toute politique de gestion de la consommation doit donc tenir compte des préférences de l'utilisateur.

2.3.2 Interface avec le système d'exploitation

La répartition de la gestion de la consommation entre le matériel, pour les mécanismes de mise en veille, et le système d'exploitation, pour l'implantation d'une politique de gestion, implique l'existence de communications bidirectionnelles entre ces deux éléments. La communication du matériel vers le système d'exploitation permet à chaque périphérique de déclarer ses capacités à contrôler sa consommation. Dans l'autre sens, elle permet au système d'exploitation de provoquer un changement de mode de fonctionnement sur un périphérique particulier suivant les décisions prises en fonction de la politique de gestion de la consommation employée. Dans les architectures de type PC[19], ces interactions ont lieu entre

[19]Personal Computer

trois acteurs : les périphériques, le BIOS[20] et le système d'exploitation. La grande variété de chacun de ces acteurs nécessite la définition d'interfaces, l'un d'elles est l'ACPI[21] [Int96].

La spécification décrite dans [Int96] utilise un ensemble de variables pour déterminer l'état de veille du système (Gx), d'un périphérique (Dx), et du processeur (Cx). L'état de veille du système étant lui-même subdivisé en plusieurs états (Sx). Chacune de ces variables peut prendre une valeur entière entre 0 (pleinement actif) et 3 (complètement arrêté). Les définitions de ces états, notamment pour les périphériques, sont suffisamment larges pour pouvoir être adaptée à tout type de périphérique. Mais cette spécification ne se limite pas à un ensemble d'états, elle définit également une machine virtuelle interne qui permet de mettre en œuvre des traitements plus complexes. Cette machine virtuelle exécute du P-code[22], et n'est utilisable que par les pilotes ACPI, mais elle permet une écriture plus générique de ces pilotes et offre la notion d'événement. La flexibilité offerte par cette machine virtuelle permet par exemple de gérer les seuils d'alerte de température, ou de décharge de la batterie, ainsi que les traitements associés.

2.3.3 Politiques d'endormissement

Pour réduire la consommation, le système d'exploitation cherche à mettre en veille tout les périphériques qui ne contribuent pas au fonctionnement actuel du système. La technique généralement utilisée est basée sur des délais de garde. Elle consiste à armer un délai de garde à la fin de l'utilisation d'un périphérique, si ce dernier est de nouveau utilisé avant l'expiration du délai, alors ce dernier est réarmé à sa valeur initiale, dans le cas contraire, c'est-à-dire si le délai expire sans que le périphérique n'ai été utilisé, alors ce dernier est mis en veille. Cette méthode des délais de garde est simple à mettre en œuvre, mais ses performances dépendent fortement des valeurs des délais de garde fixés pour chaque périphérique. L'étude menée dans [BBCR98] porte sur l'instrumentation d'un système d'exploitation afin de caractériser pour chaque périphérique les lois d'arrivée des requêtes. Les mesures ont permis de fixer les valeurs des délais de gardes, appelé MIT[23]. Des corrélations ont également été recherchées entre les activations des différents périphériques, mais bien que tous les périphériques d'entrée (permettant à l'utilisateur d'agir sur le système, c'est-à-dire le clavier et la souris) aient une forte probabilité d'agir sur l'activité du disque dur, c'est le paramètre d'auto-corrélation qui reste prédominant (si un périphérique vient d'être utilisé, il a une forte probabilité qu'il soit prochainement réutilisé).

Un des problèmes majeurs des délais de garde vient de la consommation inutile qui persiste entre la fin de l'activité du système et l'expiration du délai. Pour remédier à ce problème, on peut utiliser des algorithmes de prédiction [BDM98]. Ces algorithmes cherchent à déterminer les périodes d'inactivité du système (instant de fin de l'activité et instant de reprise de l'activité). Si l'algorithme prédit une période d'inactivité et que le système a élu la

[20] Basic Input-Output Subroutines
[21] Advanced Configuration and Power Interface Specification
[22] code exécutable par la machine virtuelle mais indépendant du processeur
[23] *Minimum Idle Time* ou temps minimum d'inactivité

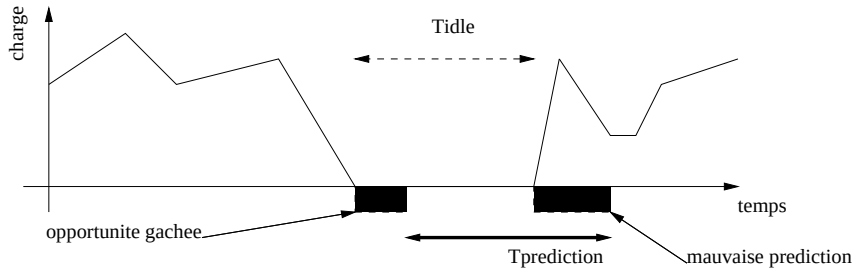

FIG. 3: Schéma d'une prédiction

tâche *idle*[24] comme tâche courante, alors le système passe immédiatement dans un mode de sommeil. La prédiction de la fin de la période d'inactivité permet d'anticiper le redémarrage du système (processus généralement assez long) afin d'être pleinement opérationnel lorsque la première tâche demandera à s'exécuter. Cette méthode permet d'éviter que la mise en sommeil ne dégrade les performances du système.

La figure 3 illustre une prédiction ainsi que les deux types d'erreurs susceptibles de ce produire avec cette technique, les opportunités gâchées et les mauvaises prédictions. Les premières apparaissent lorsqu'une période d'inactivité se présente sans que l'algorithme ne l'ait prédit, on retrouve dans ce cas la même situation qu'avec les délais de garde, il s'écoulera un certain temps entre le début de la période d'inactivité et la détection de l'erreur de la prédiction, ce qui retarde la mise en sommeil. Le second type d'erreur, les mauvaises prédictions ont lieu lorsque l'algorithme a prédit une période d'inactivité plus longue qu'elle ne l'est réellement, le mécanisme de réveil anticipé ne fonctionne alors pas, et on observe une diminution des performances.

Les prédictions de ces algorithmes ne sont généralement pas exploitées directement, mais passent d'abord dans un filtre [GBS+95] dont le but est de limiter les erreurs de prédiction. Le filtrage peut être effectué avec un simple filtre passe-bas pour éviter de prendre en compte les périodes d'inactivité de trop faible durée. Il peut également s'agir d'un seuil sur la fiabilité des prédictions. Ou bien, le filtre peut prendre en compte des paramètres de l'environnement, inconnus de l'algorithme de prédiction.

2.4 Adaptation dynamique de la vitesse du CPU

L'adaptation de la vitesse du CPU peut-être vu comme un contrôle plus fin des techniques de mises en veilles décrites précédemment. Les processeurs actuels ont des capacités de traitement de plus en plus importantes, cet accroissement de leur puissance de calcul a provoqué une augmentation significative de leur consommation : un processeur Intel 80386 ne consomme que 2 Watts alors qu'un Pentium II peut atteindre les 30 Watts [TSR+98]. Mais le fonctionnement d'un processeur n'est pas constant, il est constitué de périodes de

[24] pseudo-tâche qui correspond à une absence d'activité du processeur

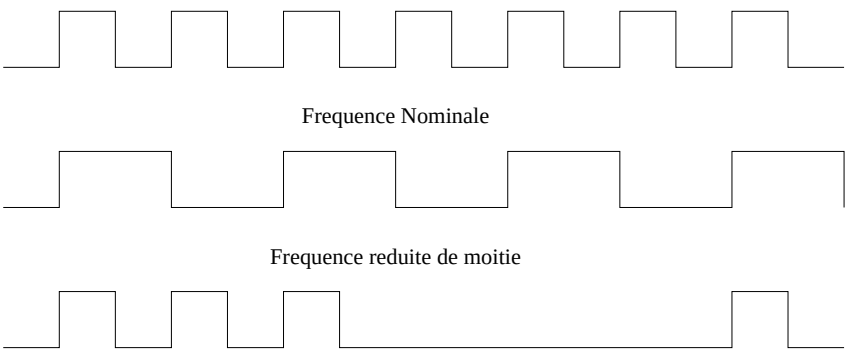

FIG. 4: Variation de fréquence d'horloge

traitement et de périodes d'inactivité. Inactivité est pris ici dans le sens où le processeur n'effectue pas un traitement utile pour l'utilisateur ou le système, mais il continue à consommer de l'énergie. Cette énergie dissipée constitue une perte pure pour l'appareil puisqu'elle ne participe en rien à la production d'un résultat. L'idée est donc d'adapter la puissance de traitement du processeur, et donc sa consommation, aux besoins actuels du système. Pour cela deux méthodes sont envisagées, l'une agit sur le signal d'horloge, et l'autre sur la tension d'alimentation du processeur.

2.4.1 Inhibition d'horloge

Cette première méthode vise donc à éviter le traitement des signaux de l'horloge pendant les périodes d'inactivité du processeur [OK98]. En effet, la consommation due à la propagation du signal d'horloge représente près de 40% de la consommation totale du processeur [TSR+98]. Lorsque la capacité de traitement du processeur est supérieure à la quantité de traitement à effectuer, il est possible diminuer la fréquence moyenne de l'horloge afin de diminuer la consommation que sa propagation induit. La figure 4 montre comment, à partir du signal d'horloge original, on peut arriver à diviser par deux la fréquence de fonctionnement d'un composant. L'une des méthodes consiste à utiliser un diviseur de fréquence afin d'obtenir un signal d'horloge d'une fréquence moitié moins élevée, l'autre consiste à activer le composant à sa fréquence nominale pendant la moitié de sa période (*burst*), et ensuite inhiber le signal d'horloge jusqu'à la fin de la période. La fréquence moyenne ainsi obtenue a alors une valeur égale à la moitié de la fréquence de base. Ces périodes d'arrêt du processeur peuvent facilement être obtenues sur certains processeurs qui possèdent une instruction spéciale pour la mise en veille, c'est notamment le cas des processeurs Pentium avec leur instruction `hlt` qui leur permet de passer en mode veille (consommation diminuée de 85% par rapport à sa valeur nominale) en moins de $50\mu sec$ [GCW95].

2.4.2 Adaptation du voltage

La diminution de la fréquence d'horloge telle que présentée ci-dessus, permet de diminuer la consommation due à la propagation inutile du signal d'horloge, mais cela ne modifie pas la consommation des cycles utiles du processeur. Cependant, s'il est possible de déterminer dynamiquement la fréquence requise pour que le processeur puisse effectuer correctement ses traitements en cours, cela offre une nouvelle possibilité pour diminuer la consommation : l'adaptation dynamique du voltage [RS98]. En effet, la fréquence de fonctionnement d'un circuit électronique est limitée par sa tension d'alimentation car c'est d'elle que dépend le temps de stabilisation des signaux après une commutation. Malheureusement les fréquences de fonctionnement actuelles de processeurs sont telles que les tensions d'alimentation doivent être relativement élevées, ce qui augmente considérablement la consommation vu que cette dernière est proportionnelle au carré de la tension d'alimentation (voir équation [1]). S'il est possible d'ajuster la fréquence de fonctionnement du processeur à la quantité de traitements en cours, et que simultanément la tension d'alimentation est diminuée de façon à être la plus faible possible tout en permettant un fonctionnement correct à cette fréquence, la consommation pourra être ajustée au plus juste pour le traitement en cours[WWDS94]. La suite de ce document porte sur cette technique en abordant plus en détails les contraintes matérielles d'un tel mécanisme et en étudiant plusieurs algorithmes déjà proposés pour obtenir une bonne adaptation de la fréquence en fonction des besoins du système.

3 Adaptation dynamique du voltage

La réalisation d'une adaptation dynamique du voltage nécessite d'une part un matériel capable d'ajuster sa fréquence et sa tension à la demande et d'autre part un algorithme capable de déterminer qu'elle doit être la fréquence de fonctionnement courante. Dans un premier temps c'est l'aspect matériel qui est abordé. Ensuite plusieurs algorithmes d'adaptation de fréquence sont étudiés, certains ne prenant en compte aucune contrainte temporelle, et d'autres cherchant à adapter la fréquence tout en respectant les contraintes de temps-réel que peuvent avoir certaines applications.

La diminution de la consommation induite par cette technique d'adaptation dynamique vient de la diminution de la tension d'alimentation, il est cependant beaucoup plus simple de réfléchir en terme de variation de fréquence. Donc dans la suite de ce chapitre, la tension d'alimentation est supposée asservie à la fréquence du processeur, toute variation de la fréquence d'un processeur entraîne une adaptation de cette tension d'alimentation de ce processeur. Ainsi les termes « variation de voltage » et « variation de fréquence » sont utilisés sans distinctions pour exprimer une adaptation coordonnée de la fréquence et de la tension d'alimentation.

3.1 Aspect matériel

La capacité d'un processeur à pouvoir modifier sa fréquence de fonctionnement requiert des composants matériels spécifiques. Notamment, il doit exister un mécanisme permettant

au processeur d'influer sur sa propre vitesse d'horloge, qui provient généralement d'un oscillateur à quartz dont la fréquence est régulée par une PLL[25]. Certaines études théoriques [WWDS94] [GCW95] [HPS98] prennent comme hypothèse qu'il est possible de faire varier la tension proportionnellement à la fréquence de manière continue entre la valeur maximale et une borne inférieure, et que les temps de transition d'un mode de fonctionnement à un autre sont négligeables. Bien que permettant de simplifier le modèle, de telles hypothèses ne sont pas très réalistes. En effet, les PLLs présentes dans le circuit de contrôle du signal d'horloge sont des boucles d'asservissement dont le temps de stabilisation est assez important [CGX96]. Même les recherches actuelles sur les régulateurs de tension variable n'arrivent pas à diminuer le temps de transition en dessous de plusieurs millisecondes par Volt [NYM97]. De plus, il faut ajouter à ce temps de transition, le délai dû à la propagation du nouveau signal d'horloge dans les autres composants, ce qui provoque au final des retards qui ne sont pas toujours acceptables dans des environnements temps-réel. Il est cependant possible de se limiter à un nombre restreint d'états de fonctionnement sans engendrer de grandes pertes au niveau des économies d'énergie [CGX96]. Le fait d'avoir un nombre fini de modes de fonctionnement permet de les caractériser avec une grande précision et de diminuer les temps de transition à quelques cycles d'horloge.

3.2 Aspect logiciel

Plusieurs études ont porté sur les algorithmes d'adaptation de la vitesse du processeur, certaines abordent le problème au niveau du système d'exploitation [GCW95], alors que d'autres essaient d'impliquer les applications dans le mécanisme de décision [CGX96]. La prise en compte des contraintes temporelles des applications complexifie la recherche d'une fréquence de fonctionnement minimale, mais plusieurs algorithmes [HPS98] [HKQ+98] proposent déjà des solutions à ce problème.

3.2.1 Algorithmes existants

Gérer l'adaptation de la vitesse du CPU au niveau du système d'exploitation revient à un problème classique : prévoir les besoins du système dans un futur proche en ne connaissant que son fonctionnement passé (et généralement seul une fraction du comportement passé est disponible). La méthode utilisée dans [WWDS94] consiste à diviser le temps en intervalles réguliers de 10 à 50 millisecondes appelés fenêtres et à évaluer la quantité de traitement effectuée dans chacune de ces fenêtres. Cette évaluation se base sur le temps passé dans la boucle *idle* (ce qui correspond à l'absence d'activité de la part des applications), ou sur le nombre de tâches n'ayant pas eu le temps de s'exécuter pendant cet intervalles (ces tâches non exécutées sont appelées *excès*). Lorsque la proportion de temps *idle* devient trop importante, le système d'exploitation diminue la vitesse du processeur, et lorsque le nombre d'excès n'est pas nul, il augmente la vitesse du processeur. La variation de la vitesse du processeur se fait de manière progressive, mais il existe un seuil pour le nombre d'excès au-delà du quel le système accélère immédiatement le processeur à sa vitesse maximale, ce qui

[25] *Phase Locking Loop*, ou boucle à verrouillage de phase

permet une réaction plus rapide aux piques de charge. L'inconvénient d'un tel algorithme vient du report des tâches d'une fenêtre de temps à une autre. En effet, les tâches en excès qui n'ont pas pu être exécutées dans une fenêtre donnée voient leur exécution reportée à la fenêtre suivante, mais sans garantie qu'elles pourront s'y exécuter. Une tâche peut ainsi être reportée de fenêtre en fenêtre jusqu'à ce que le processeur atteigne une vitesse suffisante pour exécuter toutes les tâches en attente ce qui ne permet pas de garantir le délai de terminaison d'une tâche particulière.

L'algorithme décrit dans [WWDS94] se limite à adapter la vitesse du processeur pour chaque intervalles en fonction des résultats de l'intervalles précédent, il ne peut donc pas gérer de manière satisfaisante les régimes de fonctionnement subissant de grandes variations (malgré le seuil permettant de passer le processeur à la vitesse maximale). L'étude décrite dans [GCW95] cherche à repérer des profils d'utilisation afin de mieux adapter sa réponse aux variations de charge. Pour cela les algorithmes ne se limitent pas à l'étude du dernier intervalle, mais exploitent les résultats obtenues sur plusieurs de ces intervalles. De nombreux algorithmes sont étudiés : étude de la charge moyenne, recherche de cycles, recherche de motifs, etc. Mais les meilleures performances sont obtenues avec l'algorithme PEAK qui cherche à gérer les surcharges ponctuelles. Pour cela il peut décider d'augmenter la vitesse du processeur dans un intervalle de temps très court, mais également diminuer cette vitesse en un temps tout aussi court. Mais cela est sans doutes fortement lié à la méthode de simulation utilisée. En effet, dans [GCW95] les résultats sont obtenus par simulation des algorithmes sur des traces réelles, et les traces utilisées sont celles de l'utilisation d'un éditeur de texte (Emacs). Ce type d'application interactive à effectivement besoin d'un temps de réponse très court afin de paraître réactif à l'utilisateur (d'où l'augmentation rapide de la fréquence du processeur en cas d'arrivé d'un nouveau traitement) mais son activité est constituée de traitements sporadiques fonction du comportement de l'utilisateur, les périodes d'inactivité sont donc fréquentes (d'où la diminution rapide de la fréquence du processeur à la fin d'un traitement). Dans [PBB98] la même méthode de simulation est utilisée pour évaluer quelques uns des algorithmes décrits dans [GCW95] mais les traces utilisées visent à étudier trois profils d'utilisation différents. Le premier est une application interactive (carnet d'adresses) très irrégulière car fortement dépendante des requêtes de l'utilisateur, mais ne requièrant pas une grande quantité de traitement. Le second profil représente une application de restitution d'un flux audio en haute fidélité, la quantité de traitement est assez importante, mais extrêmement régulière. Enfin le dernier profil correspond au décodage d'un flux vidéo MPEG, demandant une grande quantité de traitement mais présentant de fortes irrégularités. C'est avec le second profil, celui du flux audio, que les algorithmes atteignent leurs meilleures performances, avec une diminution de la consommation pouvant dépasser les 80% [PBB98]. Les performances obtenues avec le premier profil, l'application interactive, sont un peu moins bonnes, avec une diminution de la consommation de 30 à 50%. Et les plus mauvaises performances sont obtenues avec le décodage du flux MPEG, cas où la diminution de la consommation dépasse difficilement les 20%.

Une autre méthode pour adapter la vitesse du processeur aux traitements en cours requière la collaboration de l'application qui s'exécute. Dans [CGX96], l'application en ques-

tion effectue un décodage de trames MPEG. Les informations contenues dans ces trames ainsi qu'une bonne connaissance des caractéristiques du processus de décodage (évaluation de l'ordre du filtre à utiliser), permette de prévoir de manière suffisamment précise la quantité de traitement nécessaire pour chaque trame. L'application est alors capable pour chaque décodage de trame, d'ajuster la vitesse du processeur pour le traitement à effectuer. Il faut cependant noter que cette étude porte sur le décodage MPEG par un DSP, dans un environnement mono-application et parfaitement caractérisé. Une telle méthode serait beaucoup plus complexe à mettre en œuvre si plusieurs applications se partageaient le processeur.

3.2.2 temps-réel

Les algorithmes décrits précédemment servent à réguler la fréquence de fonctionnement d'un processeur en fonction de sa charge, mais ce problème est traité indépendamment du travail de l'ordonnancement des tâches. Dans le cas d'un environnement temps réel, le temps d'exécution des tâches est l'un des paramètres essentiels, qu'il soit utiliser directement comme dans [XP90] ou bien indirectement, comme c'est le cas dans le test de faisabilité de l'algorithme Rate Monotonic [LL73]. Or le fait de faire varier la fréquence de fonctionnement du processeur fait également varier le temps d'exécution des différentes tâches. Il est donc nécessaire d'intégrer la gestion de la fréquence à l'algorithme d'ordonnancement afin de pouvoir diminuer la consommation sans mettre en péril le respect des contraintes temporelles des applications.

Algorithme basé sur l'EDF Un algorithme [HPS98] a déjà été proposé incluant la variation de fréquence du processeur dans l'ordonnancement de tâches temps-réel. Cet algorithme se base sur l'ordonnancement EDF[26] [LL73], et gère des tâches périodiques et sporadiques. Le test d'acceptation d'une nouvelle tâche dans le système est en fait un test de faisabilité de l'ordonnancement EDF, il vérifie donc qu'à sa fréquence maximale, le processeur sera capable d'exécuter toutes les tâches. L'ordonnancement des tâches est effectuer en fonction de leurs échéances, mais un calcul supplémentaire est effectué pour fixer la fréquence de fonctionnement du processeur. Pour cela, l'ordonnanceur calcule, pour chaque tâche j, le paramètre U_j qui correspond au plus petit taux d'utilisation du processeur permettant de respecter l'échéance de cette tâche. Ce taux dépend de l'échéance de cette tâche, de son temps d'exécution ainsi que du temps d'exécution de toutes les tâches précédent cette tâche suivant l'ordonnancement EDF. Ensuite, lors de l'exécution des tâches, l'ordonnanceur fixe la fréquence de fonctionnement du processeur en fonction de la plus grande des valeurs des U_j des tâches qui restent à exécuter (la fréquence peut donc être modifiée après chaque terminaison de tâche). Cet algorithme hérite malheureusement des inconvénients de l'EDF, c'est-à-dire que toutes les tâches doivent être périodiques et indépendantes les unes des autres. De plus, parmi les hypothèses prises sur le modèle de l'architecture matérielle, on trouve que la fréquence du processeur peut-être fixée à une valeur quelconque, or les premiers processeurs commerciaux à permettre ces changements de fréquence ne supportent

[26] Earliest Deadline First

qu'un nombre limité de points de fonctionnement ayant chacun une fréquence et une tension bien déterminée [Kla00][Int99]. Un autre hypothèse utilisée est que les temps de transition d'un fréquence à une autre sont négligeable, or même si les données techniques sur les processeurs à variation de fréquences sont encore peu nombreuses, il apparaît que les temps de transitions se chiffreront au moins en dizaines de cycles processeurs [Int99]. Il est donc nécessaire d'attendre les mesures de performances réelles avant de pouvoir déterminer si ces temps sont négligeables ou non.

Algorithme basé sur les régions Une autre approche est proposée dans [HKQ+98], comme pour la méthode décrite ci-dessus, l'ordonnancement est divisé en deux phases. La première consiste à réaliser un ordonnancement des tâches. Pour cela, le temps est divisé en régions, c'est-à-dire en intervalles de temps délimités par des événements de type « arrivée d'une nouvelle tâche » ou « échéance d'une tâche ». L'algorithme d'ordonnancement attribue à chaque région une tâche à exécuter en fonction des contraintes des différentes tâches. Les contraintes des tâches sont évaluées en fonction du nombre de possibilités dont dispose l'ordonnanceur pour placer une tâche sur des régions entre son instant d'arrivée et son échéance. Plus le nombre de possibilités sera grand, plus les contraintes seront considérées comme faibles et inversement, plus le nombre de possibilités sera petit, plus les contraintes seront considérées comme fortes. L'ordonnanceur commence donc par les tâches ayant les contraintes les plus fortes pour terminer par celles ayant les contraintes les plus faibles. Pour chaque région où une tâche a été affectée, l'ordonnanceur fixe une fréquence de fonctionnement. Une fois que cette première phase est terminée, une phase d'optimisation de la consommation commence, elle consiste à ajuster les bornes et les fréquences de chaque région afin d'obtenir pour chaque nouvelle région la fréquence de fonctionnement la plus faible possible. Ces ajustements sont possible grâce au découpage du temps en régions. En effet avec un tel découpage, les tâches affectées à une région ne consomme pas forcément tout le temps disponible dans cette région, c'est en récupérant ce temps inutilisé que l'ordonnanceur peut diminuer la fréquence du processeur. Afin d'obtenir un meilleurs ordonnancement, la phase de placement des tâches sur les régions comporte un paramètre aléatoire, ce qui permet à l'ordonnanceur de générer plusieurs ordonnancements possibles à partir d'un seul algorithme. Pour chaque tirage de ce paramètre aléatoire, les deux phases de l'ordonnancement sont effectuées et au final, l'ordonnanceur choisit parmi tous les ordonnancements générés celui dont la consommation est la plus basse. L'évaluation de cet algorithme a été effectuée par simulation à partir de données constructeur sur différents processeurs, ainsi qu'un modèle de comportement des caches. Cependant cet algorithme repose sur des hypothèses similaires à celles de l'algorithme précédant : la variation de tension est supposée continue de 0.8V à 3.3V (ce qui induit une variation de fréquence continue elle aussi) et le temps de transition d'une fréquence de fonctionnement à une autre est considérée comme étant négligeable. Là encore il faudra attendre les mesures de performances réelles de ces nouveaux processeurs afin de vérifier que les hypothèses utilisées ne sont pas trop éloignées du comportement de ces derniers.

Conclusion

En résumé, nombreuses sont les techniques permettant de diminuer la consommation dans un système embarqué. Certaines de ces techniques, comme la diminution de la tension d'alimentation, les activations séparées ou bien encore les circuits reconfigurables, interviennent dès la conception des composants qui constitueront le système. Leur utilisation permet d'obtenir une diminution importante de la consommation due aux traitements effectués. Des techniques logicielles permettent une meilleurs exploitation des capacités du matériel : fonctions en lignes, déroulage de boucles, ainsi qu'une utilisation optimale des ressources externes : exécutions distantes, adaptation des protocoles de communication. Enfin, les techniques hybrides, basées sur une collaboration entre un module logiciel et des fonctionnalités matérielles, permettent d'ajuster au mieux la consommation d'un système à son niveau d'activité par des mécanismes de mise en veille et d'adaptation dynamique de tension.

Avec l'explosion du marché des systèmes embarqués grand public, ces techniques sont appelées à devenir un composant essentiel des systèmes à venir. L'utilisation de plus en plus courante d'applications multimédia entraînera également une augmentation significative de la puissance de calcul requise pour ces appareils, et introduira de nouvelles contraintes de performances. Mais au-delà de ce problème de performances, ces applications multimédia nécessitent des propriétés bien précises quant à la capacité du système à leur assurer un bon comportement temporel. Ces pourquoi les techniques de diminution de la consommation devront être de plus en plus souvent couplées à l'ordonnancement de ces systèmes.

Références

[AR96] Arthur Abnous and Jan Rabaey. Ultra-low-power domain specific multimedia processors. In *Proceedings of the* IEEE VLSI *Signal Processing Conference*, San Francisco (CA), October 1996.

[ASWR98] Arthur Abnous, Katsunori Seno, Marlene Wan, and Jan Rabaey. Evaluation of a low-power reconfigurable DSP architecture. In *Proceedings of the Reconfigurable Architectures Workshop*, Orlando (FL), March 1998.

[BBB+95] E. Brewer, T. Burd, F. Burghardt, A. Burstein, R. Doering, K. Lutz, S. Narayansaramy, T. Pering, B. Richards, T. Truman, R. Katz, J. Rabaey, and R. Brodersen. Design of wireless portable systems. In *Proceedings of the* IEEE *International Computer Society Conference (*COMPCON95*)*, pages 169–176, March 1995.

[BBCR98] Luca Benini, Alessandro Bogliolo, Stefano Cavalluci, and Bruno Riccó. Monitoring system activity for os-directed dynamic power management. In *Proceedings 1998 international symposium on Low Power Electronics and Design*, pages 185–190, 1998.

[BDM98] Luca Benini and Giovanni De Micheli. *Dynamic Power Management, Design Techniques and CAD Tools*. Kluwer Academics Publishers, 1998.

[CGX96] Anantha Chandrakasan, Vadim Gutnik, and Thucydides Xanthopoulos. Data driven signal processing: An approach for energy efficient computing. In *Proceedings of the 1996 international symposium on Low-Power Electronics and Design*, pages 347–352, 1996.

[GBS+95] Richard Golding, Peter Bosch, Carl Stealin, Tim Sullivan, and John Wilkes. Idleness is not sloth. In *Proceedings of the Winter'95* USENIX *Conference*, pages 201–222, 1995.

[GCW95] Kinshuk Govil, Edwin Chan, and Hal Wasserman. Comparing algorithms for dynamic speed-setting of a low-power CPU. In *First ACM International Conference on Mobile Computing and Networking*, 1995.

[GW94] Tom Germond and David L. Weaver. *The* SPARC *Architecture Manual - Version 9*. SPARC International Inc., 1994.

[HKQ+98] Inki Hong, Darko Kirovski, Gang Qu, Miodrag Potkonjak, and Mani B. Srivastava. Power optimization of variable voltage core-based systems. In *Proceedings of the* 35^{th} *annual conference on Design Automation Conference*, 1998.

[HPS98] Inki Hong, Miodrag Potkonjak, and Mani B. Srivastava. On-line scheduling of hard real-time tasks on variable voltage processor. In *Proceedings of the 1998 IEEE/ACM international conference on Computer-aided design*, pages 653–656, 1998.

[HWO97] Patrick Hicks, Matthew Walnock, and Robert Michael Owens. Analysis of power consumption in memory hierarchies. In *Proceedings of the 1997 international symposium on Low-Power Electronics and design*, pages 239–242, 1997.

[Int96] Intel Corporation and Microsoft Corporation and Toshiba Corporation, http://www.teleport.com/ acpi/. *Advanced Configuration and Power Interface Specification Revision 1.0*, 1996.

[Int99] Intel. Pentium iii processor mobile module: Mobile module connector 2 (mmc-2) featuring intel speedstep technology. Technical report, Intel Corporation, 1999.

[KBN95] Uming Ko, Poras T. Balsara, and Ashwini K. Nanda. Energy optimization of multi-level processor cache architectures. In *Proceedings 1995 International Symposium on Low-Power Electronics and Design*, pages 45–49, 1995.

[KK98] Robin Kravets and P. Krishnan. Power management techniques for mobile communication. In *The fourth annual ACM/IEEE International Conference on Mobile Computing and Networking*, pages 157–168, 1998.

[Kla00] Alexander Klaiber. The technology behind crusoe processors. Technical report, Transmeta Corporation White Paper, January 2000.

[LH98] Yanbing Li and Jörg Henkel. A framework for estimating and minimizing energy dissipation of embedded HW/SW systems. In *Proceedings of the* 35^{th} *annual conference on Design Automation Conference*, 1998.

[LHW97] Yann-Rue Lin, Cheng-Tsung Hwang, and Allen C.-H. Wu. Scheduling technics for variable voltage low-power design. *ACM Transactions on Design Automation of Electronics Systems*, 2(2):81–97, April 1997.

[LL73] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1), January 1973.

[LT95] Mike Tien-Chien Lee and Vivek Tiwari. A memory allocation technique for low-power embedded dsp software. In *Proceedings 1995 IEEE Symposium on Low-Power Electronics*, October 1995.

[MC95] E. Musoll and J. Cortadella. Scheduling and resource binding for low-power. In *Proceedings of the eighth international symposium on system synthesis*, pages 104–109, 1995.

[NYM97] Won Namgoong, Mengchen Yu, and Teresa Meng. A high-efficiency variable-voltage CMOS dynamic DC-DC switching regulator. In *Proceedings of the* ISSCC'97, 1997.

[OK98] Hidekiyo Ozawa and Seiya Kitagawa. Power management technology. *Fujitsu Scentific and Technical Journal*, 34(1):68–77, September 1998.

[PBB98] Trevor Pering, Tom Bird, and Robert Brodersen. The simulation and evaluation of dynamic voltage scaling algorithms. In *Proceedings 1998 International Symposium on Low-Power Electronics and Design*, pages 76–81, 1998.

[Rab97] Jan M. Rabaey. Reconfigurable processing: The solution to low-power programmable DSP. In *Proceedings 1997 ICASSP Conference*, Munich, April 1997.

[RRPK98] Alexey Rudenko, Peter Reiher, Gerald J. Popek, and Geoffrey H. Kuenning. Saving portable computer battery power through remote process execution. *Mobile Computing and Communications Review*, 2(1):19–26, January 1998.

[RRPK99] Alexey Rudenko, Peter Reiher, Gerald J. Popek, and Geoffrey H. Kuenning. The remote processing framework for portable computer power saving. In *Proceeding of the 1999* ACM *Symposium on Applied Computing*, February 1999.

[RS98] Lorch Jacob R. and Alan Jay Smith. Software strategies for portable computer energy management. IEEE *Personal Communications Magazine*, 5(3):60–73, June 1998.

[SD95] Ching-Long Su and Alvin M. Despain. Cache designs for energy efficiency. In *Proceedings of the* 28^{th} *Hawaï international Conference on System Science*, January 1995.

[SRL90] Lui Sha, Ragunathan Rajkumar, and John P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9), September 1990.

[TMW94] Vivek Tiwari, Sharad Malik, and Andrew Wolfe. Power analysis of embedded software: A first step towards software power minimization. IEEE *Transactions on* VLSI *Systems*, December 1994.

[TSR+98] Vivek Tiwari, Deo Singh, Suresh Rajgopal, Gaurav Mehta, Rakes Patel, and Franklin Baez. Reducing power in high-performance microprocessors. In *Proceedings of the* 35^{th} *annual conference on Design Automation Conference*, pages 732–737, 1998.

[WWDS94] Mark Weiser, Brent Welch, Alan Demers, and Scott Shenker. Scheduling for reduced CPU energy. In *Proceedings of the First Symposium on Operating System Design and Implementation*, November 1994.

[XP90] Jia Xu and David Lorge Parnas. Scheduling processes with release times, deadlines, precedence , and exclusion relations. IEEE *Transactions on Software Engineering*, 16(3):360–369, March 1990.

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399