Réduction de la consommation statique des systèmes temps-réel multiprocesseurs

Vincent Legout, Mathieu Jan
CEA, LIST,
Laboratoire Fondements des Systèmes Temps Réels Embarqués
F-91191 Gif-sur-Yvette, France
{vincent.legout,mathieu.jan}@cea.fr

Laurent Pautet
Institut Telecom
TELECOM ParisTech
LTCI - UMR 5141 Paris, France
laurent.pautet@telecom-paristech.fr

***Résumé*—Réduire la consommation énergétique – composée des consommations dynamique et statique – des systèmes embarqués est aujourd'hui une préoccupation majeure pour leurs concepteurs. Dans cette optique, cet article introduit un algorithme d'ordonnancement temps-réel multiprocesseur visant d'abord à réduire la consommation statique, devenue prépondérante, dans la consommation énergétique totale. L'objectif est d'utiliser les états basse-consommation des processeurs où le processeur est inactif et sa consommation considérablement réduite. Notre approche consiste à créer de larges périodes d'inactivité pour utiliser les états basse-consommation les plus profonds et diminuer le nombre de réveils nécessaires pour retourner d'un état basse-consommation à l'état nominal. Pour ce faire, le temps processeur inutilisé est modélisé comme une tâche supplémentaire et la programmation linéaire est utilisée pour réduire son nombre de préemptions. Des simulations montrent que notre solution est plus efficace énergétiquement que les solutions existantes.**

I. Introduction

Réduire la consommation énergétique des systèmes temps-réel embarqués présente plusieurs avantages, notamment d'augmenter la durée de vie limitée des systèmes fonctionnant sur batteries. Cet effort énergétique s'inscrit également dans la tendance générale à réduire plus globalement notre consommation énergétique.

Cet article se concentre sur les processeurs qui sont les plus importants consommateurs d'énergie. Leur consommation énergétique est divisée en deux : la consommation dynamique et la consommation statique. La première est dépendante de l'activité du processeur tandis que la seconde est essentiellement due au courant de fuite et ne varie pas quelque soit l'activité du processeur. Au niveau logiciel, deux solutions existent pour réduire la consommation et s'intéressent soit à la consommation statique soit à la consommation dynamique. Pour la consommation dynamique, la technique DVFS (*Dynamic Voltage and Frequency Scaling*) permet de réduire la fréquence du processeur tandis que la technique DPM (*Dynamic Power Management*) cherche à utiliser les états basse-consommation du processeur, seule solution pour limiter le courant de fuite. Les travaux existants se sont presque exclusivement concentrés sur DVFS ([7], [5]) car la consommation dynamique était prépondérante. DPM n'était alors utilisé qu'après DVFS. Cependant cette hypothèse n'est plus valable avec les processeurs actuels ; la consommation statique étant maintenant majoritaire [8], il est nécessaire d'exploiter en premier lieu DPM puis éventuellement DVFS.

La contribution de cet article est LPDPM (Linear Programming DPM), un algorithme d'ordonnancement optimal multiprocesseur global ayant pour objectif de réduire la consommation statique des systèmes temps-réel. Un système temps-réel a des contraintes temporelles strictes qui ne peuvent être violées, par conséquent l'utilisation des états basse-consommation doit être réfléchie pour permettre au processeur de retrouver l'état nominal à temps de sorte qu'aucune échéance ne soit violée. LPDPM élargit les périodes d'inactivité des processeurs pour permettre l'utilisation d'états basse-consommation plus économes en énergie. Hors-ligne, l'utilisation totale est connue en utilisant le pire temps d'exécution (WCET) de chaque tâche. L'objectif peut donc également être exprimé comme la réduction du nombre de périodes d'inactivité.

A. État de l'art

Les algorithmes d'ordonnancement multiprocesseurs ayant pour objectif de réduire la consommation énergétique utilisent majoritairement un ordonnancement partitionné où la migration des tâches entre processeurs est interdite. Ces solutions ([5], [13], [6]) ne peuvent par conséquent être optimales du point de vue de l'utilisation des ressources. Utiliser le partitionnement empêche également le regroupement des périodes d'inactivité de chaque processeur. Pour remédier à ce problème, [13] et [6] assignent un processeur à chaque tâche hors-ligne mais permettent néanmoins en-ligne, et sous certaines conditions, aux travaux d'être exécutés sur d'autres processeurs afin d'étendre des périodes d'inactivité.

[3] est la seule solution utilisant une approche globale. Leur objectif est de n'utiliser qu'un seul processeur et de n'activer les processeurs restants qu'uniquement lorsqu'une échéance est sur le point d'être violée. Étant basé sur EDF, leur algorithme d'ordonnancement n'est pas optimal. De plus, chaque décision d'ordonnancement nécessite un calcul en $O(n^3)$ pour retarder les futures exécutions sans violer d'échéances ce qui rend l'algorithme non utilisable en pratique (n étant le nombre de tâches).

Ces solutions utilisent DVFS ou DPM mais se basent sur la priorité des tâches – qu'elle soit statique ou dynamique – pour prendre leurs décisions. Nous pensons qu'un algorithme

FIGURE 1. Ordonnancement avec Global-EDF.

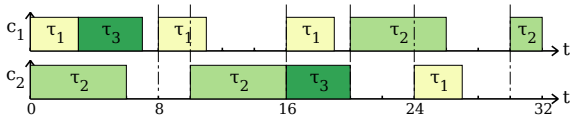
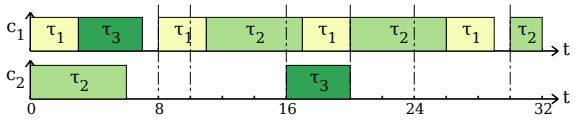


FIGURE 2. Autre ordonnancement possible.



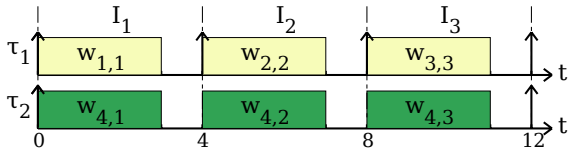
capable de prendre une décision d'ordonnancement en fonction de l'état global du système est supérieur à un algorithme basé sur l'évaluation d'un critère tâche par tâche (c.-à.-d. de manière indépendante) pour optimiser la consommation statique. Pour illustrer cette affirmation, la figure 1 représente l'ordonnancement avec Global-EDF d'un système composé de 2 processeurs c_1 et c_2 et d'un ensemble de 3 tâches τ_1, τ_2 et τ_3 dont les WCET et périodes sont respectivement (3,8), (6,10) et (4,16). Cet algorithme ordonnance en priorité les tâches dont les échéances sont les plus proches. Sur cet exemple, 7 périodes d'inactivité sont générées, alors que sur la figure 2 l'ordonnancement obtenu ne présente que 3 périodes d'inactivité. Par exemple, dans ce dernier ordonnancement, l'exécution de τ_2 à $t = 10$ est retardée mais pas à $t = 20$. Cette variation dans la prise de décision tient au fait que l'algorithme utilisé s'appuie sur l'état global du système, contrairement à ce qui est montré par la figure 1. La contribution de cet article est de proposer un algorithme d'ordonnancement qui permet d'obtenir ces gains en consommation statique.

II. MODÈLE

Nous faisons l'hypothèse que le système possède m processeurs homogènes et doit ordonnancer un ensemble Γ de n tâches indépendantes, préemptibles, synchrones et périodiques. Nous utilisons une approche globale où toutes les tâches peuvent migrer entre les différents processeurs. Pour permettre l'utilisation de DPM, chaque processeur possède au minimum un état basse-consommation où il est inactif et où sa consommation est réduite. Un délai est nécessaire pour passer d'un état basse-consommation à l'état nominal et plus l'état basse-consommation est économe en énergie, plus ce délai est important. Pour choisir l'état basse-consommation le plus adapté, le système doit connaître la durée de la période d'inactivité.

Chaque tâche τ libère un travail à chaque période T. Chaque travail j est caractérisé par son WCET $j.c$ et son échéance. Nous supposons que l'échéance d'un travail est égale à la date de libération du prochain travail de la même tâche (i.e. ensemble de tâches à échéances implicites). L'hyperpériode de l'ensemble de tâches est nommé H et est le PPCM de toutes les périodes des tâches de Γ. L'utilisation u_i d'une tâche i est le ratio entre son WCET et sa période. L'utilisation globale de l'ensemble de tâches est $U = \sum_{i=0}^{n-1} u_i$. Pour éviter la situation où tous les processeurs sont actifs, nous faisons l'hypothèse

FIGURE 3. Exemple avec 2 tâches de période 4 et 12.



que l'utilisation globale est telle que $U \in]m-1; m[$. Si $U < m-1$, il est possible de retrouver sur cette hypothèse en laissant un ou plusieurs processeurs au repos.

Nous utilisation la modélisation proposée par [9] où l'hyperpériode est divisée en intervalles. Les frontières des intervalles sont les périodes des tâches. I est l'ensemble des intervalles et $|I_k|$ est la durée de l'intervalle k. Un travail peut être présent sur plusieurs intervalles. Nous notons $w_{j,k}$ le poids du travail j sur l'intervalle k. Le poids d'un travail représente la fraction de processeur utilisable par ce travail dans cet intervalle. J_k est l'ensemble contenant tous les travaux présents sur l'intervalle k. E_j est l'ensemble des intervalles où le travail j est présent, E_j doit contenir au moins un intervalle.

L'ensemble de tâches de la figure 3 est composé de 2 tâches τ_1 et τ_2 et de 4 travaux (les travaux 1 à 3 appartiennent à la tâche τ_1 et le travail 4 appartient à la tâche τ_2). Sur cet exemple, $E_4 = \{1, 2, 3\}$ et $J_1 = \{1, 4\}$.

L'ordonnancabilité de l'ensemble de tâches peut être exprimée sous la forme d'un programme linéaire où les équations suivantes doivent être vérifiées [9] :

$$\forall k, \sum_{j \in J_k} w_{j,k} \leq m \tag{1}$$

$$\forall k, \forall j, 0 \leq w_{j,k} \leq 1 \tag{2}$$

$$\forall j, \sum_{k \in E_j} w_{j,k} \times |I_k| = j.c \tag{3}$$

La première inégalité signifie que l'utilisation sur un intervalle ne doit pas être supérieure au nombre de processeurs. La seconde inégalité interdit au poids d'un travail d'être supérieur à 1 pour que ce travail soit ordonnançable sur un intervalle. La dernière équation garantie que le temps d'exécution de chaque travail est égal à son WCET.

Résoudre le programme linéaire composé des équations (1), (2) et (3) permet d'obtenir le poids de chaque travail sur chaque intervalle et donc un ordonnancement valide. L'objectif de la prochaine section est ensuite d'optimiser cet ordonnancement afin d'élargir les périodes d'inactivité.

III. CONTRIBUTION

Hors-ligne, nous faisons l'hypothèse que chaque tâche utilise toujours son WCET. Donc quelque soit l'algorithme d'ordonnancement utilisé, l'utilisation est constante sur une hyperpériode. Le nombre de périodes d'inactivité peut donc être utilisé pour évaluer l'efficacité énergétique. Plus ce nombre est faible, meilleur est l'algorithme car : 1) réveiller un processeur

entraîne une pénalité énergétique et 2) cela permet d'obtenir des périodes d'inactivité plus larges, et donc d'activer des états basse-consommation plus économe.

Afin d'exprimer notre problème, nous modélisons le temps d'inactivité à l'aide d'une nouvelle tâche τ'. Cette tâche est également une tâche périodique dont la période est égale à H et l'utilisation à $m - U$. τ' n'a donc qu'un seul travail sur une hyperpériode. L'ajout de cette tâche τ' est possible car l'utilisation globale est comprise entre $m-1$ et m. Ainsi, l'utilisation de τ' est strictement inférieure à 1 et τ' est ordonnançable. Le nouvel ensemble de tâche a une utilisation égale à m. Pour alléger les notations, nous notons w_k le poids de la tâche τ' sur l'intervalle k.

À noter que 2 processeurs ne peuvent être inactifs simultanément car la tâche τ' ne peut être active sur deux processeurs en même temps – propriété également valable pour les autres tâches. A noter que la tâche τ' ne représente que le temps d'inactivité hors-ligne. En-ligne, certaines tâches peuvent ne pas utiliser leur WCET et plusieurs processeurs peuvent être inactifs simultanément. Avec l'ajout de cette tâche τ', notre objectif d'obtenir un minimum du nombre de périodes d'inactivité peut être reformulé en la minimisation de nombre de préemptions de τ' sur une hyperpériode.

Pour réduire le nombre de préemptions de τ', nous procédons en deux étapes. La première étape est d'éviter les préemptions à l'intérieur des intervalles et la seconde est d'éviter les préemptions entre intervalles.

Pour éviter les préemptions à l'intérieur des intervalles, le poids de la tâche τ' doit être soit 0 soit 1 dans chaque intervalle. Pour exprimer cette objectif, nous introduisons deux variables binaire f_k et e_k tel que :

$$f_k = \begin{cases} 0 & \text{if } w_k = 1 \\ 1 & \text{otherwise} \end{cases} \tag{4}$$

$$e_k = \begin{cases} 0 & \text{if } w_k = 0 \\ 1 & \text{otherwise} \end{cases} \tag{5}$$

f_k représente les intervalles où $w_k = 1$ (i.e. *full*) tandis que e_k représente les intervalles où $w_k = 0$ (i.e. *empty*). Ainsi, la fonction objectif minimisant $\sum_k f_k + e_k$ permet de minimiser le nombre d'intervalles où $0 < w_k < 1$.

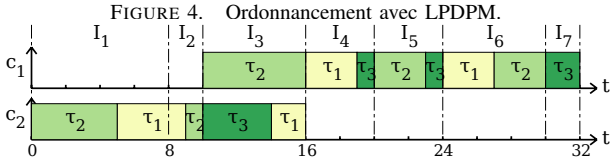
Ensuite, pour réduire le nombre de préemptions entre intervalles, nous introduisons deux variables binaires supplémentaires fc_k et ec_k tel que :

$$fc_k = \begin{cases} 1 & \text{if } f_k = 1 \text{ and } f_{k+1} = 0 \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

$$ec_k = \begin{cases} 1 & \text{if } e_k = 1 \text{ and } e_{k+1} = 0 \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

Minimiser $\sum_k fc_k$ permet de faire en sorte que les intervalles où $w_k = 1$ soient consécutif. Le même raisonnement est utilisé pour ec_k pour les intervalles où $w_k = 0$. L'objectif final de notre programme linéaire (MILP) est :

FIGURE 4. Ordonnancement avec LPDPM.



$$\text{Minimize} \sum_k f_k + e_k + fc_k + ec_k \tag{8}$$

En-ligne, à l'intérieur des intervalles, l'ordonnancement des tâches se fait en utilisant le poids de chaque tâche en utilisant par exemple l'algorithme optimal IZL [10]. L'état basse-consommation le plus adapté est activé lors de chaque exécution de τ'. L'ordonnancement des autres tâches impacte la consommation énergétique uniquement lorsque le poids de τ' est différent de 0 ou 1. Pour ces intervalles, notre ordonnanceur fait en sorte que τ' soit ordonnancée en début ou fin d'intervalle pour coller l'exécution de τ' à une autre exécution de l'intervalle précédent ou suivant. Ceci afin d'élargir les périodes d'inactivités. Enfin, lorsque ces périodes d'inactivités ne peuvent pas être utilisées pour exploiter des états basse-consommation, l'utilisation de la technique DVFS peut être envisagée. Par manque de place, nous ne pouvons donner plus de détails sur la cette partie de notre ordonnanceur.

La figure 4 représente l'ordonnancement LPDPM de l'ensemble de tâche de la figure 1. Alors que Global-EDF générait 9 périodes d'inactivité, LPDPM n'en crée que 2. Sur cet exemple, le poids de la tâche τ' est de 1 sur l'intervalle 3 et de 0 sur tous les autres intervalles. A noter que pour équilibrer la charge, le processeur exécutant τ' est mis à jour à chaque exécution de τ'.

IV. ÉVALUATION

Pour comparer notre solution aux algorithmes existants, nous avons utilisé un simulateur en se basant sur les caractéristiques des processeurs STM32L (ARM Cortex-M3 [2]) et FreeScale PowerQUICC (e.g. MPC551x [1]). Les 4 états basse-consommation sont détaillés dans le tableau I. A l'aide de ce simulateur, nous avons généré des ensembles de tâches aléatoires et les avons ordonnancés sur 2 hyperpériodes. Cette simulation utilise 4 processeurs et chaque ensemble de tâches est composé de 10 tâches. Chaque tâche a une utilisation comprise entre 0.01 et 0.99 générée de manière aléatoire en utilisant l'algorithme UUniFast [4]. La période de chaque tâche est choisie aléatoirement entre 10 et 100.

Pour 10 utilisations globale entre 3 et 4, 2000 ensembles de tâches ont été générés et ordonnancés avec RUN [12], U-EDF [11], IZL (i.e. MILP + IZL in [10]) et LPDPM. RUN, U-EDF et IZL sont 3 algorithmes d'ordonnancement multiprocesseurs optimaux dont le but est de réduire le nombre de préemptions et de migrations. Il n'existe en effet aucun autre algorithme multiprocesseur optimal dont l'objectif est de réduire la consommation statique. Les implémentations de LPDPM et d'IZL utilisent CPLEX pour résoudre le programme linéaire. Le temps laissé à CPLEX pour trouver une

FIGURE 5. Nombre moyen de périodes d'inactivité.

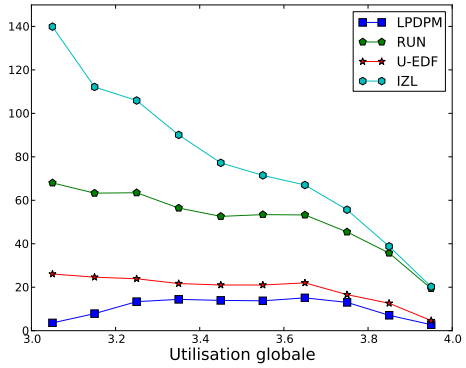
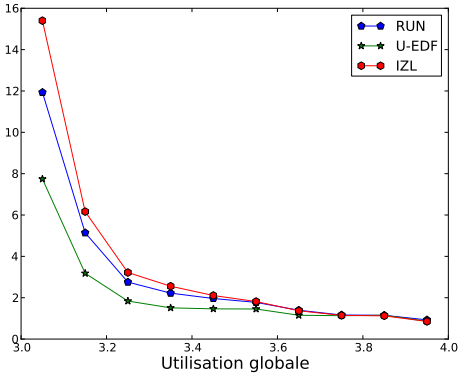


FIGURE 6. Consommation globale relative (LPDPM = 1).



solution est de 1 minute, ce qui s'est avéré suffisant pour obtenir une solution même non-optimale.

TABLE I
ÉTATS BASSE-CONSOMMATION UTILISÉS POUR LA SIMULATION.

Mode	Consommation (normalisée)	Délai
Run	1	
Sleep	0.01	$0.01ms$
Stop	0.001	$1ms$
Standby	0.00001	$10ms$

La figure 5 détaille le nombre moyen de périodes d'inactivité pour chaque ordonnanceur suivant l'utilisation globale. LPDPM génère toujours moins de périodes d'inactivité que les autres ordonnanceurs, les gains de LPDPM étant plus important lorsque l'utilisation globale est proche de 3. En effet, plus le temps d'inactivité est important, plus il est facile à LPDPM de regrouper les périodes d'inactivité et le gain par rapport aux autres solutions augmente.

La consommation moyenne relativement à LPDPM pour chaque utilisation globale est donnée en figure 6. La consommation de LPDPM est toujours 1 et cette consommation représente la consommation lorsqu'aucune tâche n'est active. La consommation lors de l'exécution des tâches étant identique pour tous les algorithmes d'ordonnancement, la différence entre ces différentes solutions se fait exclusivement lorsque les processeurs sont inactifs (i.e. lorsque la tâche τ' est en cours d'exécution pour LPDPM). Lorsque l'utilisation globale est faible, LPDPM est jusqu'à 10 fois plus efficace que les autres solutions, et cette différence se réduit lorsque l'utilisation globale augmente car le temps d'inactivité utilisable par les états basse-consommation est réduit.

Nous avons également calculé le nombre de préemptions et de migrations engendrées sur l'ensemble des tâches par les 4 solutions. Sans avoir encore ajouté d'optimisations à LPDPM pour réduire les préemptions et migrations des tâches autre que τ', les résultats de LPDPM sont similaires à ceux de U-EDF, et toujours mieux de 1.4 fois ceux de RUN et IZL.

V. CONCLUSION

Cet article présente LPDPM, un algorithme d'ordonnancement temps-réel optimal réduisant la consommation statique des systèmes multiprocesseurs. Il augmente la taille des périodes d'inactivité pour permettre l'utilisation d'états basse-consommation plus économes en énergie. Les simulations effectuées montrent que le résultat est atteint, que ce soit par rapport au nombre de périodes d'inactivités ou à la consommation globale. Nous sommes en train d'étendre cette étude aux systèmes à criticité mixte et aux systèmes hétérogènes.

RÉFÉRENCES

[1] *FreeScale MPC5510 Microcontroller Family Reference Manual.*

[2] *ST Microelectronics. STM32L151xx and STM32L152xx advanced ARM-based 32-bit MCUs. Reference Manual RM0038, 2011.*

[3] M. Bhatti, M. Farooq, C. Belleudy, and M. Auguin. Controlling energy profile of rt multiprocessor systems by anticipating workload at runtime. In *SYMPosium en Architectures nouvelles de machines*, 2009.

[4] E. Bini and G. C. Buttazzo. Biasing effects in schedulability measures. In *Proc. of the 16th Euromicro Conf. on Real-Time Systems*, pages 196–203, 2004.

[5] J.-J. Chen, H.-R. Hsu, and T.-W. Kuo. Leakage-aware energy-efficient scheduling of real-time tasks in multiprocessor systems. In *Proc. of the 12th IEEE Real-Time and Embedded Technology and Applications Symp.*, pages 408–417, 2006.

[6] H. Huang, F. Xia, J. Wang, S. Lei, and G. Wu. Leakage-aware reallocation for periodic real-time tasks on multicore processors. In *Proc. of the 2010 Fifth Intl. Conf. on Frontier of Computer Science and Technology*, pages 85–91, 2010.

[7] R. Jejurikar, C. Pereira, and R. Gupta. Leakage aware dynamic voltage scaling for real-time embedded systems. In *Proceedings of the 41st annual Design Automation Conference*, pages 275–280, 2004.

[8] E. Le Sueur and G. Heiser. Dynamic voltage and frequency scaling : The laws of diminishing returns. In *Proc. of the 2010 Workshop on Power Aware Computing and Systems (HotPower'10)*, pages 1–8, 2010.

[9] M. Lemerre, V. David, C. Aussaguès, and G. Vidal-Naquet. Equivalence between schedule representations : Theory and applications. In *Proceedings of the 2008 IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 237–247, 2008.

[10] T. Megel, R. Sirdey, and V. David. Minimizing task preemptions and migrations in multiprocessor optimal real-time schedules. In *Proc. of the 31st IEEE Real-Time Systems Symp.*, pages 37–46, 2010.

[11] G. Nelissen, V. Berten, J. Goossens, and D. Milojevic. Reducing preemptions and migrations in real-time multiprocessor scheduling algorithms by releasing the fairness. In *Proc. of the 17th Intl. Conf. on Embedded and Real-Time Computing Systems and Applications*, pages 15 –24, 2011.

[12] P. Regnier, G. Lima, E. Massa, G. Levin, and S. Brandt. Run : Optimal multiprocessor real-time scheduling via reduction to uniprocessor. In *Proc. of the IEEE 32nd Real-Time Systems Symp.*, pages 104–115, 2011.

[13] E. Seo, J. Jeong, S. Park, and J. Lee. Energy efficient scheduling of real-time tasks on multicore processors. *IEEE Trans. Parallel Distrib. Syst.*, 19(11) :1540–1552, 2008.