Ordonnancement temps réel avec profils variables de consommation d'énergie

S. Jeannenot **P. Richard** **F. Ridouard**
{pascal.richard@ensma.fr} {frederic.ridouard@ensma.fr}

Laboratoire d'Informatique Scientifique et Industrielle
École Nationale de Mécanique et d'Aérotechnique
Téléport 2 – BP 40109 F-86961 Chasseneuil Futuroscope Cedex, France

Résumé : Les systèmes embarqués intègrent de plus en plus des fonctions de calcul nécessitant l'usage de processeur performant à faible consommation d'énergie. Pour répondre à cette double exigence, des processeurs pouvant changer dynamiquement leur fréquence de fonctionnement ont été développés comme l'Intel StrongArm et le Crusoe de Transmeta. Nous présentons des algorithmes d'ordonnancement de tâches périodiques soumises à des échéances impératives qui s'exécuteront sur un processeur à vitesse variable. Nous considérons les processeurs dont la vitesse varie dynamiquement dans le temps mais possédant un nombre fini de paliers de vitesse. Les tâches n'ont pas des profils identiques de consommation d'énergie, afin de représenter l'utilisation différente des ressources par les tâches. Nous nous limitons aux solutions où chaque tâche est allouée à une vitesse constante durant toute son exécution.

Plan

1- Introduction
2- Etat de l'art
3- Heuristiques pour les problèmes à vitesse discrète
4- Expérimentations
5- Conclusion

Mots clés : Temps-réel, Ordonnancement, Processeur à vitesse variable, Système embarqué.

1 Introduction

Les systèmes embarqués intègrent de plus en plus de fonctionnalités nécessitant une puissance de calcul importante. Toutefois, le fonctionnement de tels systèmes repose sur des batteries. La minimisation de l'énergie consommée par le système devient alors un critère très important. Les techniques proposées dans ce but reposent sur le matériel et son architecture comme l'utilisation des processeurs à vitesse variable, ou sur des techniques logicielles comme la mise en veille des composants non utilisés et enfin sur des techniques hybrides combinant les deux approches précédentes [6, 11].

Dans la suite nous considérons des processeurs permettant d'adapter dynamiquement leur fréquence de fonctionnement à la quantité de travail à traiter, comme l'Intel StrongArm et le Crusoe de Transmeta. Ces processeurs permettent d'ajuster conjointement la fréquence de l'horloge et la tension d'alimentation du processeur afin de réduire l'énergie consommée en quelques micro-secondes. Pour une fréquence donnée, la tension d'alimentation minimisant l'énergie consommée peut être déterminée [6]. La fonction de consommation d'énergie est usuellement quadratique en fonction de la vitesse du processeur (dans tous les cas elle est convexe). Nous supposerons que la vitesse de travail du processeur est bornée par deux constantes S_{min} et S_{max}.

Un logiciel temps réel est défini par un ensemble de tâches périodiques ou apériodiques. Dans la suite, une tâche périodique τ_i est définie par sa durée d'exécution en nombre de cycles processeurs C_i, sa période entre deux activations successives T_i. L'échéance d'une instance de tâches survient D_i unités de temps après son réveil. Une tâche τ_i est à échéance sur requête si son échéance relative D_i est égale à la période T_i. Les tâches périodiques sont supposées ci-après être à démarrage simultané. Dans le cas de tâches apériodiques, une tâche τ_i est définie par sa date d'arrivée r_i, son temps d'exécution en nombre de cycles du processeur C_i et son échéance absolue d_i.

L'ordonnancement de tâches sur des processeurs à vitesse variable génère de nombreux travaux dans la littérature. L'ordonnanceur ne se limite pas à définir l'ordre des tâches à exécuter par le processeur. Il doit en plus définir, à

chaque instant, la vitesse du processeur. La prise en compte de cette nouvelle dimension généralise la problématique de l'ordonnancement temps réel. Deux paramètres émergent pour classifier les problèmes étudiés dans la littérature :

- les vitesses évoluent de façon continue ou discrète dans le temps. Dans le cas continu, nous supposons que les vitesses appartiennent à l'intervalle $[S_{min},S_{max}]$ et dans le cas discret, il existe m différents paliers de vitesse : $\{S_1,\ldots,S_m\}$. A notre connaissance, tous les processeurs existants utilisent un nombre fini de paliers de vitesse.
- les tâches ont des profils de consommation identiques ou variables. Lorsque le profil est identique, les tâches utilisent la même quantité d'énergie par unité de temps. Afin de tenir compte plus précisément de l'activité des tâches utilisant des ressources matérielles différentes (comme l'utilisation d'un DSP par exemple), le profil variable de consommation d'énergie permet de définir pour chaque tâche sa propre consommation d'énergie par unité de temps.

Dans le paragraphe 2, nous passons rapidement en revue les résultats connus pour toutes les combinaisons de ces deux paramètres. Le troisième paragraphe présente des heuristiques lorsque les tâches ont des profils variables de consommation d'énergie et que les processeurs ont un nombre fini de paliers de vitesse. Le paragraphe 4 présente le résultat des expérimentations numériques.

2 Etat de l'art

2.1 Vitesse continue et profils de consommation identiques

Yao *et al.* [14] proposent un algorithme polynomial permettant d'ordonnancer des tâches apériodiques afin de minimiser l'énergie consommée. Les tâches sont ordonnancées selon la règle Earliest Deadline First (EDF), qui est optimale pour construire une séquence de tâches respectant leurs échéances. Le principe de l'algorithme est de définir un *intervalle critique* dans l'ordonnancement comme l'intervalle ayant la plus grande charge de travail. Celui-ci débute nécessairement par l'arrivée d'une tâche et se termine nécessairement par une échéance d'une tâche. *Les tâches critiques* sont les tâches dont la date

d'activation et l'échéance appartiennent à l'intervalle critique. *L'intensité* d'un intervalle est définie par la somme des durées des tâches critiques. L'énergie sera minimisée sur cette portion d'ordonnancement en fixant la vitesse du processeur égale à la durée de l'intervalle divisée par la charge processeur dans cet intervalle. Avec cette vitesse, le processeur n'est jamais oisif dans l'intervalle critique. Les tâches ainsi ordonnancées sont supprimées du problème ; les dates de réveils et les échéances des tâches restantes sont ajustées de façon à ne pouvoir empiéter sur l'intervalle critique précédemment ordonnancé. Un nouvel intervalle critique est alors déterminé et le même principe est appliqué itérativement tant que toutes les tâches n'ont pas été traitées. On constate que la vitesse de processeur ne change pas durant l'exécution d'une tâche et que les changements de vitesse n'interviennent que de façon discrète dans le temps. Une implémentation simple de cet algorithme conduit à une complexité en $O(n^2)$.

Récemment, Gaujal *et al.* [7] ont proposé un algorithme en $O(n \log n)$ pour résoudre le même problème. Cet algorithme est fondé sur la recherche d'un plus court chemin dans un graphe. Cet algorithme, tout comme celui de [14], peut être appliqué à des configurations de tâches périodiques en considérant l'ensemble des instances de tâches réveillées durant une hyperpériode.

Lorsque les tâches sont à priorité fixe et apériodiques, le problème d'ordonnancement avec vitesse variable est un problème $\mathcal{NP}$-Difficile [15]. Ces auteurs proposent un algorithme polynomial hors-ligne avec des garanties de performance (le ratio entre l'énergie consommée par l'algorithme approché et celle consommée par l'algorithme optimal est borné par une constante indépendante des données du problème). Par ailleurs, dans [12] est proposée une heuristique polynomiale pour les tâches périodiques.

Dans le cas de tâches périodiques, à démarrage simultané et possédant des échéances relatives égales à leurs périodes, alors Aydin montre le résultat suivant :

Théorème 1 : *[1] La vitesse optimale du processeur qui minimise la consom-*

mation totale d'énergie en respectant toutes les échéances des tâches est constante et égale à $\overline{S} = max\{S_{min},U\}$ *où* U *est le facteur d'utilisation du processeur lorsque celui-ci fonctionne à vitesse maximale* $U = \sum_{i=1}^{n} C_i/(T_i S_{max})$

La preuve de ce résultat est donnée dans [1] et repose sur les multiplicateurs de Lagrange. Nous donnons ci-après une preuve très simple de ce résultat en analysant le comportement de l'algorithme de Yao *et al* sur ce type de configurations de tâches. Nous montrons ci-après que l'hyperpériode est un intervalle critique. En conséquence, l'algorithme de Yao *et al* ne nécessitera qu'une itération car toutes les exécutions des tâches auront alors été simultanément considérées. La vitesse sera égale à la durée de l'hyperpériode divisée par la charge des tâches sur cet intervalle. Ceci est exactement le facteur d'utilisation du processeur. La vitesse du processeur correspond alors au résultat du théorème 1. Il nous reste ainsi à montrer que l'hyperpériode est effectivement un intervalle critique pour compléter la preuve.

Propriété 1 *Lorsque l'algorithme de Yao et al. [14] est appliqué sur des tâches périodiques à échéance sur requête et à démarrage simultané, alors l'hyperpériode est un intervalle critique dès la première itération.*

Preuve : Puisque les tâches sont à démarrage simultané, le premier intervalle critique, défini par l'algorithme de Yao, *et al* débute à la date 0. Celui-ci se terminera par une échéance d'une tâche quelconque τ_k. Considérons un intervalle critique I, défini par a périodes consécutives de τ_k. Le nombre d'activations de la tâche $\tau_j,k \leq j$ dans l'intervalle I vaut :

$$\left\lfloor \frac{aT_k}{T_j} \right\rfloor$$

La durée cumulée des tâches dans l'intervalle I vaut :

$$\sum_{j=1}^{n} \left\lfloor \frac{aT_k}{T_j} \right\rfloor C_j$$

donc la charge processeur sur I est donnée par

$$U(I) = \frac{\sum_{j=1}^{n} \left\lfloor \frac{aT_k}{T_j} \right\rfloor C_j}{aT_k}$$

La charge processeur sur l'hyperpériode de longueur $H = ppcm(T_j)$ est définie par le facteur d'utilisation $U = \sum_{j=1}^{n} C_j/T_j$. Or

$$\left\lfloor \frac{aT_k}{T_j} \right\rfloor \leq \frac{H}{T_j} \qquad \forall a, \forall j$$

donc $U(I) \leq U$ et en conséquence l'hyperpériode est un intervalle critique. □.

2.2 Vitesse continue et profils de consommation variables

Aydin [1] a montré qu'il est toujours possible de définir un ordonnancement respectant les échéances tel que chaque tâche τ_i s'exécute à une vitesse constante S_i. La durée d'exécution de τ_i est alors définie par C_i/S_i. Dans le cas de tâches périodiques, le facteur d'utilisation du processeur est $\sum_{i=1}^{n} \frac{C_i}{S_i T_i}$.
Aydin *et al.* [4, 2] montrent le lien du problème de minimisation d'énergie avec celui de la maximisation de la valeur acquise pour l'exécution de partie optionnelle de tâche (reward based scheduling) [3]. Les algorithmes sont polynomiaux et reposent sur la programmation mathématique non linéaire.

2.3 Vitesse discrète et profils de consommation identiques

Gaujal *et al.* [7] propose un algorithme polynomial qui minimise le nombre de changements de fréquence d'horloge. Les tâches considérées sont apériodiques. Cet algorithme transforme la solution optimale obtenue dans le cas continu en une solution discrète avec la même consommation d'énergie. La solution discrète obtenue est donc optimale. Les auteurs proposent de plus une solution discrète minimisant le nombre de changements de vitesse.

2.4 Vitesse discrète et profils de consommation variables

Ce problème discret peut se modéliser comme une extension du problème de sac-à-dos. Cette formulation a été proposée dans [10, 9, 13]. Considérons un

processeur avec m paliers de vitesse. Soit e_{ij} l'énergie consommée par la tâche τ_i lorsqu'elle s'exécute à la vitesse S_j du processeur et soit $u_{ij} = \frac{C_i}{T_i S_j}$ la charge processeur correspondante. On définit l'affectation d'une tâche τ_i à la vitesse j par la variable bivalente $x_{ij} = 1$, pour toutes les autres valeurs de $k \neq j$ on vérifie alors $x_{ik} = 0$. La formulation du problème est alors un programme mathématique linéaire avec variables bivalentes :

$$
\begin{array}{ll}
Min & \displaystyle\sum_{i=1}^{n}\sum_{j=1}^{m} e_{ij} x_{ij} \\
 & \text{sous les contraintes} \\
 & \displaystyle\sum_{j=1}^{m} x_{ij} = 1 \qquad 1 \leq i \leq n \\
 & \displaystyle\sum_{i=1}^{n}\sum_{j=1}^{m} u_{ij} \leq 1 \\
 & x_{ij} \in \{0,1\} \qquad 1 \leq i \leq n, 1 \leq j \leq m
\end{array}
$$

La première contrainte impose que chaque tâche soit affectée à un palier de vitesse tandis que la seconde assure que le facteur d'utilisation du processeur soit inférieur ou égal à 1 (test d'ordonnançabilité pour EDF). Bien que le programme mathématique soit linéaire, sa résolution est un problème $\mathcal{NP}$-Difficile au sens faible [13].

Il est important de remarquer que ce programme linéaire avec variables bivalentes n'est pas optimal puisqu'il suppose que chaque tâche s'exécutera toujours à la vitesse constante. A notre connaissance, il n'a pas été démontré que l'ensemble des ordonnancements allouant une vitesse constante par tâche contient nécessairement une solution minimisant l'énergie totale (i.e., ensemble dominant). Toutefois, dans la suite nous nous limiterons aux ordonnancements affectant une vitesse constante pour chaque tâche. Nous pensons que cette hypothèse est réaliste puisque, comme l'illustreront les expérimentations numériques, avec un nombre important de paliers de vitesse, la solution discrète est proche de la solution continue pour laquelle l'hypothèse permet de conduire à une solution optimale.

Bien qu'un solveur généraliste puisse être utilisé pour résoudre ce programme mathématique avec variables bivalentes (comme CPLEX d'ILOG ou bien OSL

d'IBM), nous ne pouvons pas définir de durée maximum en temps de calcul ni de taille maximum de problèmes (en nombre de tâches) qui pourront être résolus. L'unique moyen de garantir de telles propriétés est de définir des heuristiques. De plus, la programmation mathématique avec variables bivalentes est un problème NP-difficle au sens fort, alors que notre problème peut, a priori, être résolu en temps pseudo-polynomial. L'utilisation d'un solveur n'est donc pas spécialement adaptée à notre problème.

3 Heuristiques pour les problèmes à vitesses discrètes et profils variables de consommation d'énergie

Nous présentons trois heuristiques de résolution du problème d'ordonnancement de tâches périodiques, à départ simultané et à échéance sur requête $(D_i = T_i)$. L'ordonnancement optimal sera donc obtenu en ordonnançant les tâches selon l'ordre EDF (Earliest Deadline First). Des expérimentations numériques seront ensuite présentées. Nous supposons que les fréquence du processeur appartiennent à l'intervalle compris entre deux valeurs $F_{min} = 300Mhz$ et $F_{max} = 833Mhz$, et que l'intervalle entre deux fréquences successives est donnée par S_{step} (par exemple 33MHz pour le processeur Crusoe de Transmeta). Les fonctions de puissance utilisées sont obtenues par régression quadratique à partir des points de fonctionnement du processeur Crusoe de Transmeta (TM5500) à plus ou moins 10% près [8]. La figure 1 donne le tube de puissance délimitant l'amplitude maximum des consommations des tâches. Dans cette figure, la vitesse est divisée par la vitesse maximale du processeur (i.e., $S_{max} = 1$). Le tube est défini par deux régressions polynomiales sur les points de fonctionnement du processor Crusoe de Transmeta à plus ou moins 10% sur les deux paramètres de régression [8].

3.1 L'algorithme du "palier constant"

Nous présentons une heuristique affectant la même vitesse de fonctionnement à toutes les tâches. L'intérêt de cette méthode est d'éviter le changement de vitesse dynamique au sein de l'application temps réel. Dans le cas continu, le théorème 1 permet de choisir la vitesse optimale identique pour toutes les

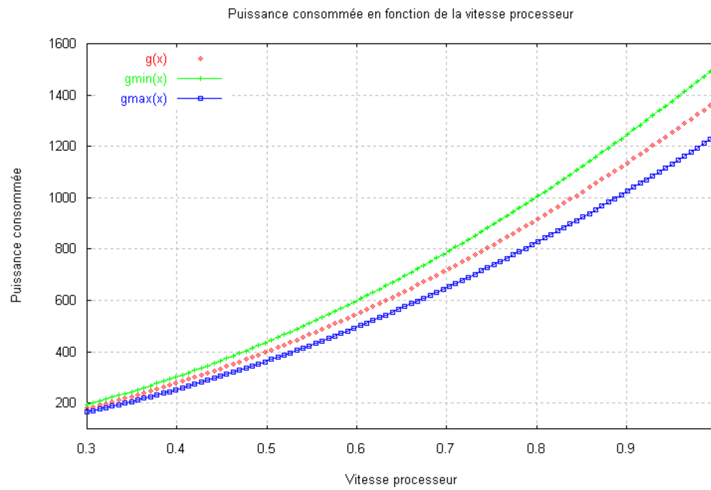

FIG. 1 – *Tube de puissance définissant les consommations des tâches*

tâches minimisant la consommation totale en énergie. Nous adaptons cette idée au cas discret en affectant toutes les tâches au palier le plus proche de la valeur idéale établie dans le cas continu.

Palier constant :
La vitesse du processeur est constante et égale à :

$$\overline{S} = S_{min} + k * S_{step}$$
$$\text{où } k = min\left\{\left\lceil\frac{U - S_{min}}{S_{step}}\right\rceil ; \left\lfloor\frac{S_{max} - S_{min}}{S_{step}}\right\rfloor\right\} \text{ et } S_{step} = \text{constante}$$

3.2 L'algorithme de la "descente en cascade"

Le principe de l'algorithme est de fixer en premier lieu toutes les tâches à la vitesse maximale du processeur. Ensuite les tâches sont classées par ordre décroissant de "saut d'énergie". Un "saut d'énergie" est égal à la différence d'énergie entre l'exécution de la tâche à la vitesse courante (par exemple pour le palier de vitesse i) et celle à la vitesse inférieure (pour le palier $i - 1$).

Si cette modification de la solution courante ne permet pas de respecter une charge processeur inférieure ou égale à 1, alors la transformation est rejetée et la tâche suivante est considérée. Le traitement s'arrête lorsque, à une itération donnée, les n tâches ont été parcourues sans avoir pu réduire leur vitesse.

3.3 L'algorithme du "recuit simulé"

Le recuit simulé est né d'une analogie entre l'optimisation combinatoire et la thermodynamique. En optimisation combinatoire, le but est de minimiser une fonction en évitant les minima locaux, alors que le nombre de solutions réalisables est souvent très élevé. Cet algorithme s'inspire des méthodes de descente, mais au lieu de rejeter une transformation entraînant une augmentation de la fonction objectif que l'on veut minimiser, on l'accepte avec une certaine probabilité [5]. La température est un paramètre de contrôle (par analogie avec le recuit métallurgique) qui rendra l'acceptation des transformations désavantageuses de moins en moins probables. Le principe de l'algorithme consiste à effectuer une transformation simple passant de la solution X_i à X_j ; on calcule la variation $\Delta f_{ij} = f(X_j) - f(X_i)$ de la fonction objectif f, et on accepte la transformation avec la probabilité $P(i,j)$ définie par :

$$\begin{aligned} P(i,j) &= 1 && \text{si } \Delta f_{ij} \leq 0 \\ P(i,j) &= e^{\frac{-\Delta f_{ij}}{t}} && \text{si } \Delta f_{ij} > 0 \end{aligned}$$

Dans cette expression, t est un paramètre de contrôle : quand la température t est élevée (initialement), un grand nombre de transformations seront acceptées ; quand t aura atteint une valeur basse, proche de zéro, seules les transformations avantageuses seront retenues, et les remontées n'auront plus lieu.

Pour résoudre le problème d'ordonnancement à vitesse variable, nous mémorisons dans une solution X_i les valeurs des vitesses des tâches (i.e. un vecteur de dimension n). Le recuit simulé fait intervenir six paramètres, que l'on doit choisir judicieusement de manière à obtenir une solution acceptable en un temps de calcul raisonnable [5]. Notre implémentation repose sur les para-

mètres suivants :

- La solution initiale est calculée par l'algorithme de "Descente en cascade".
- La température initiale est calculée à partir du maximum des Δf et du taux d'acceptation α. Δf est ici égal à la différence d'énergie lors du passage de toutes les tâches de la vitesse minimale à la vitesse maximale. On obtient ainsi une borne maximale de Δf. Etant donné que Δf peut prendre des valeurs élevées, on choisit, en contrepartie, un taux d'acceptation faible, fixé à $\alpha = 0{,}3$. En conséquence la température initiale est fixée à l'aide de la formule classique [5]: $t = -\frac{\Delta f}{\ln \alpha}$.
- La décroissance de la température est calculée par une suite géométrique de raison $\mu = 0{,}95$ permettant une décroissance rapide de la température (la température à l'itération $p + 1$ est définie par celle à l'itération p comme $t_{p+1} = \mu * t_p$)
- Le nombre de changements de température au cours de l'algorithme est constant pour tout le plan d'expérimentations et vaut 60. Cette valeur est couramment utilisée dans les recuits simulés [5].
- Le nombre de transformations élémentaires à température fixée est choisi proportionnel au carré du nombre de tâches [5]. Par exemple, pour 5 tâches, ce nombre vaut 25.
- Le voisinage correspond à l'ensemble des solutions admissibles. Le choix d'une solution dans le voisinage fait ici appel à des tirages de nombres aléatoires dans une distribution uniforme. Dans le cas présent, le voisinage est engendré par la modification de la vitesse d'une seule tâche tirée au hasard parmi l'ensemble des tâches. Sa vitesse est alors modifiée après tirage d'un nombre choisi aléatoirement dans l'intervalle $[0; 1]$: la vitesse est augmentée (palier supérieur de vitesse) si ce nombre est supérieur ou égal à 0,75 ou que la vitesse correspond au palier de vitesse minimale. Dans le cas contraire, la vitesse est réduite. On favorise donc la réduction de la vitesse des tâches, vu que le changement de vitesse n'est pas équiprobable. Une fois la solution du voisinage obtenue, on vérifie qu'elle est admissible, c'est à dire que la charge processeur totale est inférieure ou égale à 1. Si ce n'est pas le cas, on retire une solution dans le voisinage.

4 Expérimentations

Nous présentons les résultats des expérimentations numériques. Nous détaillons dans un premier temps un exemple complet permettant de comprendre le fonctionnement des trois heuristiques et de les comparés avec une solution optimale (obtenue par programmation mathématique). Dans un deuxième temps, nous avons comparé les trois heuristiques entre elles pour étudier les performances relatives.

4.1 Un exemple complet

A titre d'exemple, nous présentons les résultats numériques d'un problème avec un processeur disposant de 5 paliers de vitesse et chargé d'exécuter 5 tâches, ayant des profils différents de consommation d'énergie. La configuration de tâches est donnée dans le tableau 1. La consommation énergétique des tâches pour chaque paliers de vitesse est donnée dans le tableau 2. Ces valeurs proviennent des fonctions énergétiques tirées au hasard dans le tube de puissance présenté figure 1. Enfin, le récapitulatif des résultats obtenus est donné dans le tableau 3. L'optimum a été calculé à l'aide d'un solveur de programme mathématique avec des variables bivalentes. La figure 2 donne les résultats détaillés du recuit simulé (i.e. paliers initiaux, ceux obtenus après exécution, la fréquence d'apparition de la solution finale après 60 itérations). La figure 3 compare entre eux les résultats des trois heuristiques et de la résolution par programmation mathématique.

τ_i	r_i	C_i	D_i	T_i
1	0	6	60	60
2	0	10	80	80
3	0	11	80	80
4	0	13	90	90
5	0	6	50	50

TAB. 1 – *Paramètres des tâches*

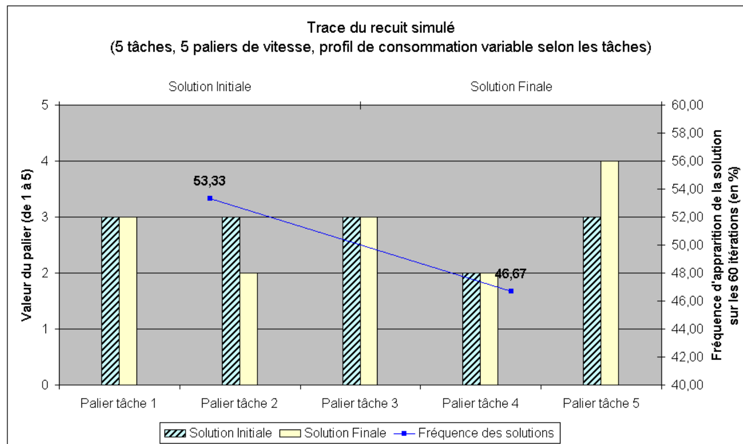

FIG. 2 – *"Trace" du recuit simulé*

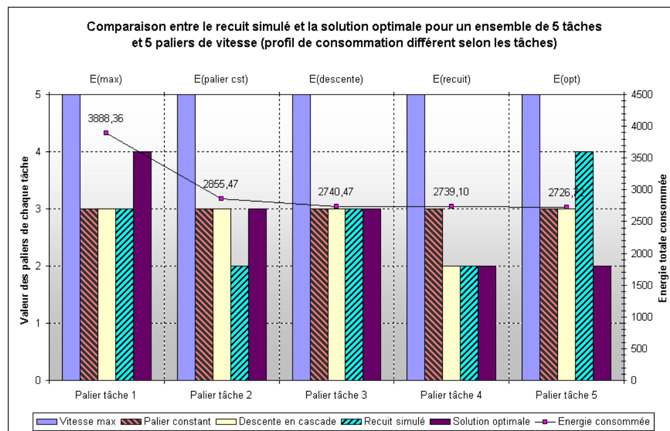

FIG. 3 – *Récapitulatif des résultats de l'exemple*

τ_i	S_1	S_2	S_3	S_4	S_5
1	301,13	375,84	455,46	537,25	620,21
2	376,41	469,80	569,32	671,57	775,26
3	414,05	516,78	626,26	738,72	852,79
4	434,96	542,88	657,88	776,03	895,86
5	361,35	451,01	546,55	644,71	744,25

TAB. 2 – *Energie consommée*

Algorithme	Choix des paliers	Charges Processeurs	Energie totale
Vitesse Max	[5;5;5;5;5]	0,626944	3888,36
Palier Constant	[3;3;3;3;3]	0,911919	2855,47
Descente Cascade	[3;3;3;2;3]	0,973714	2740,47
Recuit Simulé	[3;2;3;2;4]	0,994866	2739,1
Optimum[a]	[4;3;3;2;2]	0,998115	2726,72

TAB. 3 – *Résultats*

[a] Au sein des ordonnancements affectant une vitesse constante à chaque tâche pour toutes ses instances.

4.2 Résultats numériques

Pour chaque taille de problème (i.e., nombres de tâches et de paliers fixés), nous avons généré aléatoirement 25 instances pour obtenir des résultats statistiques satisfaisants. Le nombre de tâches a été pris dans l'ensemble $\{3,5,10,15\}$, et le nombre de paliers de vitesse varie de 3 à 15, avec un pas de 1. Nous avons mesuré deux critères :

- La charge processeur totale en fonction du nombre de paliers de vitesse.
- La déviation des heuristiques par rapport à la meilleure d'entre elles : pour chaque instance du problème, on mesure l'écart entre la solution obtenue par chaque heuristique et la meilleure solution trouvée. Ensuite, on calcule la moyenne de cet écart sur les 25 instances. La formule est

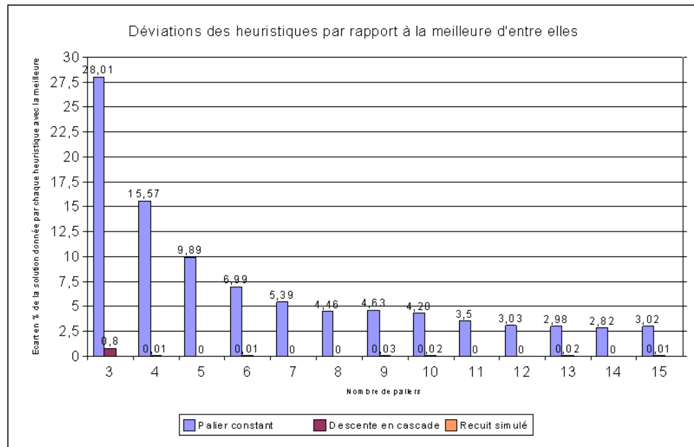

FIG. 4 – *Ecart entre les heuristiques et la meilleure pour 10 tâches*

la suivante :

$$moyenne \left(\left| \frac{E(H) - E(meilleure)}{E(meilleure)} \right| * 100 \right)$$

Dans un premier temps nous montrons le détail des résultats obtenus pour 10 et 15 tâches, puis dans un deuxième temps nous présenterons des résultats de synthèse en ne supposant fixé que le nombre de tâches (i.e., les résultats synthétisent ceux obtenus avec tous les paliers de vitesse, pour un nombre de tâches donné). L'ensemble des résultats sont présentés dans [8].

4.2.1 Résultats détaillés pour 10 et 15 tâches

Les figures 4 et 5 montrent que l'écart entre les heuristiques se ressert dès lors que le nombre de paliers de vitesse augmente. Comme l'illustre ces figures, au delà de 5 paliers de vitesse, l'heuristique "palier constant" dévie de moins de 10% des résultats du recuit simulé. L'algorithme de "descente en cascade" obtient de très bons resultats comparativement au recuit simulé.

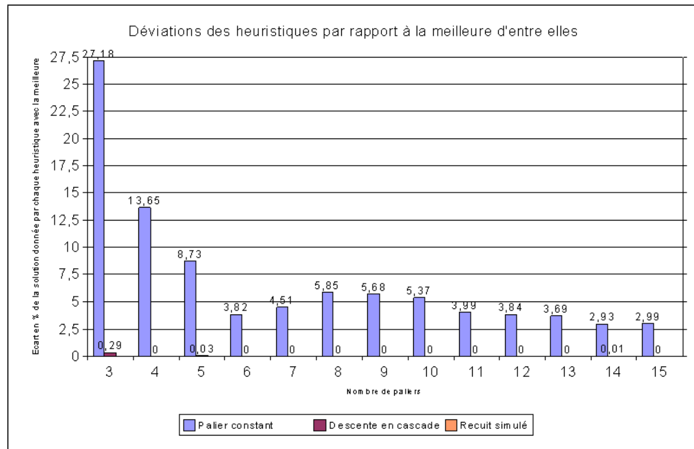

FIG. 5 – *Ecart entre les heuristiques et la meilleure pour 15 tâches*

4.2.2 Synthèse des résultats

La déviation du recuit simulé est toujours nulle car le recuit est soit la meilleure des trois heuristiques, soit n'améliore pas la solution initiale (solution fournie par l'heuristique de Descente en Cascade). Nous avons comparé les heuristiques avec la solution consistant à utiliser la vitesse maximum pour toutes les tâches. La figure 6 donne les charges processeurs obtenues en moyenne. Nous listons ci-après les conclusions quant au comportement des heuristiques.

- Heuristique du "Palier Constant" : Quel que soit le nombre de tâches, nous observons qu'elle est de plus en plus performante plus le nombre de paliers augmente. Ceci est normal, car plus l'intervalle des vitesses du processeur est discrétisé, plus le modèle continu est approché. La figure 7 montre que la déviation par rapport à la meilleure heuristique reste inférieure à 10% (figure 7). Ainsi, pour les systèmes qui ne sont pas soumis à des contraintes énergétiques trop importantes, un processeur à vitesse constante pourra être utilisé.
- Heuristique de la "Descente en Cascade" : cette heuristique permet de calculer rapidement une solution discrète proche de l'optimale. En effet,

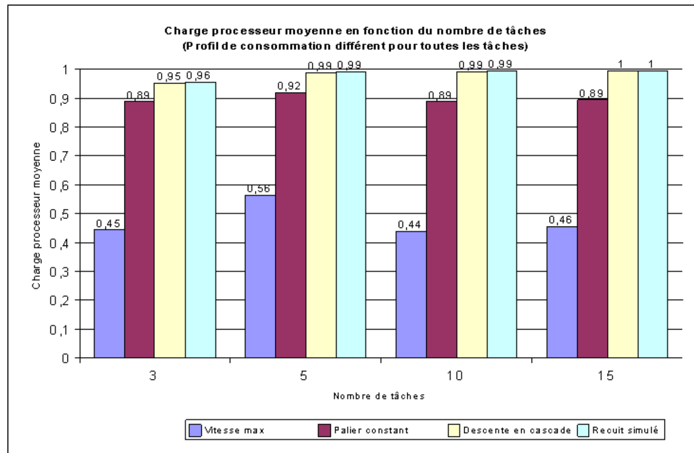

FIG. 6 – *Charge processeur moyenne*

la figure 7 montre que la déviation par rapport à la meilleure heuristique est toujours inférieure à 1%.

- Heuristique du "Recuit Simulé" : Le recuit n'améliore ainsi la solution initiale que dans 5 à 15 % des cas traités (figure 8). Ce faible taux de réussite est certainement dû, d'une part aux choix des paramètres du recuit, et d'autre part la solution initiale est déjà très proche de la solution recherchée (voire de l'optimum). Le recuit apporte alors une amélioration lorsqu'il est nécessaire de faire remonter l'énergie consommée (i.e., la fonction objectif) au cours de la recherche de la solution. Or, la recherche d'une solution se fait par le choix aléatoire dans un voisinage, et parfois la meilleure solution n'est pas atteinte dans le temps imparti. Nous pensons que lorsque le recuit améliore la solution initiale, il trouve en général la solution proche de l'optimum global. Pour approfondir l'étude de la qualité du recuit simulé, il faudrait calculer la solution optimale systématiquement pour chaque expérience à partir des données enregistrées au cours du plan d'expérimentations.

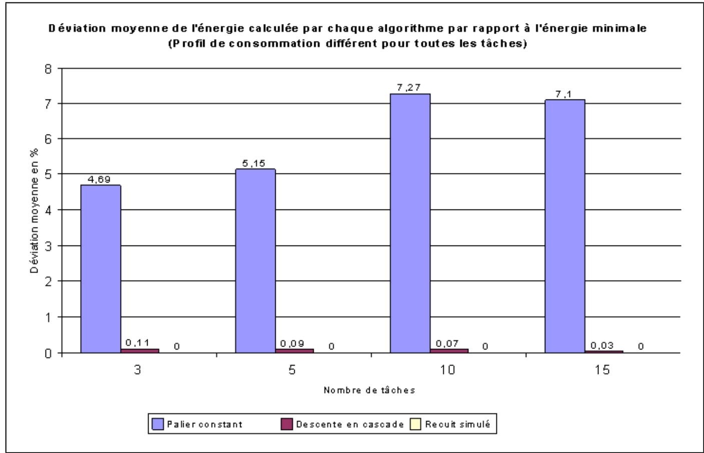

FIG. 7 – *Déviation moyenne des heuristiques*

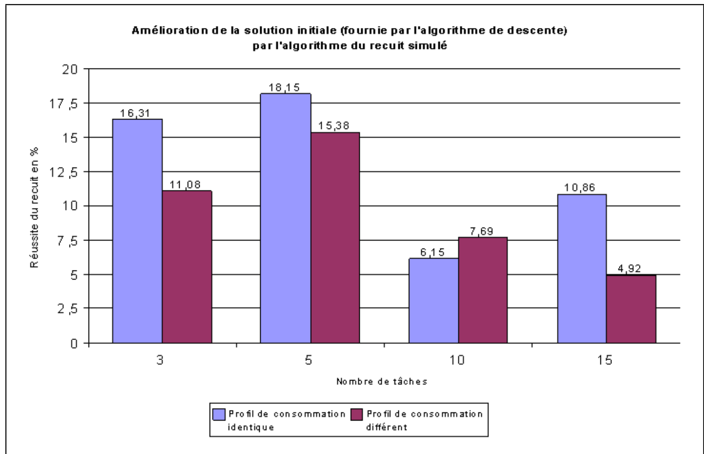

FIG. 8 – *Aperçu de la qualité du recuit simulé*

5 Conclusion

Nous avons présenté trois heuristiques de résolution du problème d'ordonnancement avec minimisation d'énergie lorsque les vitesses sont discrètes et que les profils de consommation des tâches ne sont pas identiques (i.e., variables). Nous avons expérimenté ces algorithmes ainsi que la résolution par programmation linéaire sur un exemple à 5 tâches et 5 paliers de vitesse. Puis, nous avons mené une expérimentation sur des instances de problèmes générées aléatoirement. L'ensemble des résultats montre que le recuit simulé conduit à une très bonne solution vis à vis de l'optimum, mais aussi que l'algorithme de descente en cascade fournit des résultats qui ne sont pas systématiquement améliorés par la méthode de recuit simulé. Cette dernière est donc une heuristique très intéressante en pratique.
Les perspectives de ce travail sont de considérer des configurations de tâches où l'ordonnançabilité n'est pas testée en temps linéaire. Une autre voie est de considérer simultanément l'attribution de vitesses et de priorités fixes aux tâches afin de minimiser l'énergie consommée dans un système à priorité fixe.

Références

[1] AYDIN, H. *Enhancing Performance and Fault Tolerance in Reward-Based Scheduling*. PhD thesis, University of Pittsburgh, Août 2001. pages 53–58.

[2] AYDIN, H., MELHEM, R., AND AD P.M. ALVAREZ, D. M. Power-aware scheduling for periodic real-time tasks. *IEEE Transactions on Computers, to appear*.

[3] AYDIN, H., MELHEM, R., AND AD P.M. ALVAREZ, D. M. Optimal reward-based scheduling for periodic real-time tasks. *IEEE Transactions on Computers 50*, 2 (2001), 111–130.

[4] AYDIN, H., MELHEM, R., MOSSÉ, D., AND ALVAREZ, P. Dynamic and aggressive scheduling techniques for power-aware real-time systems. *proc. Real-Time Systems Symposium (RTSS'01)* (2001).

[5] CHARON, I., GERMA, A., AND HUDRY, O. *Méthode d'optimisation combinatoire*. Masson, Paris, 1996.

[6] GAUJAL, B., AND NAVET, N. Ordonnancement sous contrainte de temps et d'énergie. In *Actes de l'école d'été temps réel (ETR'03)* (9-12 Septembre 2003), pp. 263–276.

[7] GAUJAL, B., NAVET, N., AND WALSH, C. A linear algorithm for real-time scheduling with optimal energy use. *INRIA Research report*, 4886 (July 2003).

[8] JEANNENOT, S. Ordonnancement temps réel avec contrainte de consommation d'énergie dans les systèmes embarqués. *Rapport de stage de DEA T3iA, ENSMA et Université de Poitiers* (2003).

[9] MEJIA-ALVAREZ, P., LEVNER, E., AND MOSSE, D. An integrated heuristic approach to power aware real-time scheduling. In *Workshop on Power-Aware Computer Systems (PACS'02), LNCS 2325, Springer Verlag* (2002).

[10] MEJIA-ALVAREZ, P., LEVNER, E., AND MOSSE, D. Power-optimized scheduling server for real-time tasks. In *Eighth IEEE Real-Time and Embedded Technology and Applications Symposium* (25-27 Septembre 2002), pp. 239–253.

[11] PARAIN, F., BANATRE, M., CABILIIC, G., HIGUERA, T., ISSARNY, V., AND LSEOT, J. Tenchiques de réduction de la consommation dans les systèmes embarqués temps réel. *INRIA Research report*, 3932 (May 2000).

[12] QUAN, G., AND HU, X. Energy efficient fixed priority scheduling for real-time systems on variable voltage processors. *proc. Design Automation Conference* (2001), 828–833.

[13] RUSU, C., MELHELM, R., AND MOSSE, D. Maximizing the system value while satisfying time and energy constraint. In *proc Real-Time Systems Symposium (RTSS'02)* (2002).

[14] YOA, F., DEMERS, A., AND SHENKER, S. A scheduling model for reduced cpu energy. *proc. 36th Symposium on Foundations of Computer Science (FOCS'95)* (1995), 374–382.

[15] YUN, H., AND KIM, J. On energy-optimal off-line scehduling for fixed-priority real-time systems on a variable voltage processor. *ACM Tranactions on Embedded Computing Systems 2*, 3 (2002), 393–430.