Recherche systématique pour l'ordonnancement temps réel global multiprocesseur

Olivier Buffet et Liliana Cucu-Grosjean

INRIA – 615, rue du Jardin Botanique
54600, Villers-lès-Nancy
{olivier.buffet, liliana.cucu}@inria.fr

Mots-clés : *ordonnancement temps réel, multiprocesseur, recherche systématique*

1 Introduction

Les systèmes temps réel sont généralement embarqués et ils interagissent avec l'environnement. La principale contrainte de ces systèmes concerne leur sûreté de fonctionnement et nous nous intéressons ici surtout au respect des contraintes de temps dans un contexte multiprocesseur. Dans ce contexte on différencie deux types d'approches. Les approches partitionées considèrent que toutes les instances d'une tâche sont ordonnancées sur le même processeur. Les approches globales considèrent que l'instance d'une tâche peut migrer d'un processeur à un autre processeur. Notre travail s'intéresse aux approches globales.

2 Présentation du problème

Nous considérons l'ordonnancement préemptif des tâches périodiques sur un ensemble de processeurs identiques. Un ensemble de tâches τ contient n tâches et chaque tâche est décrite par (O_i, C_i, D_i, T_i) où :

O_i est le réveil de la première instance de τ_i ;
C_i est la durée d'exécution pire cas de τ_i ;
T_i est la *période* de τ_i ;
D_i est l'*échéance* de τ_i, i.e., chaque instance générée à $O_i + (k-1)T_i$ doit finir son exécution avant $D_i + O_i + (k-1)T_i$.

On considère le cas $D_i \leq T_i, \forall i \leq n$. Le temps étant discret, tous les paramètres ont des valeurs entières.

Une solution de notre problème pour un ensemble de tâches $\tau = \{\tau_1, \ldots, \tau_n\}$ et un ensemble de m processeurs $\{P_1, \ldots, P_m\}$ est définie par un *ordonnancement* $\sigma : \mathbb{N} \to \{0, 1, \ldots, n\}^m$ où $\sigma(t) = (\sigma_1(t), \sigma_2(t), \ldots, \sigma_m(t))$ avec

$$\sigma_j(t) = \begin{cases} 0, & \text{si aucune tâche est ordonnancée sur } P_j \\ & \text{à l'instant } t; \\ i, & \text{si } \tau_i \text{ est ordonnancée sur } P_j \text{ à l'instant } t; \end{cases} \qquad \forall 1 \leq j \leq m.$$

Nous considérons le parallélisme interdit, i.e., une tâche est ordonnancée à un instant donné au plus sur un processeur.

3 Solutions proposées

Pour résoudre ce problème d'ordonnancement, nous considérons deux algorithmes de recherche systématique : (i) l'un effectue une recherche dans l'espace des ordres de priorité (fixes) sur les tâches, (ii) l'autre effectue une recherche dans l'espace des ordonnancements complets.

Recherche de priorités fixes De nombreux travaux en ordonnancement temps-réel s'intéressent aux solutions écrites sous la forme de priorités fixes indiquant dans quel ordre affecter les tâches aux processeurs.

Nous nous intéressons ici à la recherche de telles priorités fournissant un ordonnancement faisable. On représente les priorités par la liste ordonnée des tâches. L'algorithme consiste alors en une recherche en profondeur d'abord qui construit progressivement cette liste (i) en partant d'une liste vide et (ii) en ajoutant progressivement des tâches en queue de liste. Au fur et à mesure de cette recherche, on construit en parallèle l'ordonnancement détaillé induit par ces priorités (sur une hyperpériode T_H), ce qui permet d'élaguer une branche dès qu'une nouvelle tâche ne peut être ordonnancée. Un point clef est d'essayer les tâches dans un ordre astucieux pour vite trouver une solution. On observe que considérer les tâches avec une valeur $D - C$ grande en premier est très efficace, et fournit une solution sans retour arrière dans de nombreux cas. Prouver qu'un problème n'est pas faisable est par contre très coûteux.

Recherche d'un ordonnancement complet Il est possible qu'un problème soit ordonnançable mais qu'il n'existe pas de solution sous forme de priorités fixes. Une méthode sûre pour trouver une solution dans un cas faisable est d'essayer tous les ordonnancements possibles. On va pour cela formaliser le problème comme un problème de satisfaction de contraintes (CSP) et proposer un algorithme de résolution.

Notre problème d'ordonnancement temps réel multiprocesseur peut se modéliser comme un CSP avec une variable n-aire $x_j(t)$ par processeur j et pas de temps t—où $t \in 1..T_H$—indiquant quelle tâche τ_i (-1 si aucune) tourne sur le processeur j au temps t. Avec ces variables notre problème est défini par les contraintes suivantes :

$$x_j(t) \neq i,\ \forall t \notin I_{i,1} \cup \cdots \cup I_{i,\frac{T_H}{T_i}}; \tag{1}$$

$$x_j(t) = x_{j'}(t) \Leftrightarrow x_j(t) = -1; \tag{2}$$

$$\sum_{t \in I_{i,k}} \sum_{j} \delta(i, x_j(t)) = C_i,\ \forall k \in 1..\frac{T_H}{T_i}; \tag{3}$$

où $\delta(a, b) = 1$ si $a = b, 0$ si $a \neq b$.

L'algorithme proposé est ici aussi une recherche en profondeur d'abord, les variables étant ordonnées selon t d'abord, puis j. La recherche est rendue efficace entre autres (i) en ajoutant des contraintes (par exemple liés à la laxité) qui permettent de détecter un échec plus précocément, et (ii) en classant les tâches selon une heuristique informative.

On obtient un algorithme qui trouve des ordonnancements très vite, mais prend du temps à prouver l'infaisabilité d'un problème. Pour réduire le coût de ces retours arrières, nous proposons une solution consistant à focaliser la recherche sur la zone temporelle "suspecte".

Travaux existants Une approche similaire a été déjà proposée pour l'ordonnancement temps réel multiprocesseur mais dans le cas partitioné [2]. A notre connaissance l'utilisation des algorithmes de recherche systématique est une nouvelle direction de recherche initiée par [1] qui se concentre sur les techniques de programmation par contraintes.

Références

[1] L. Cucu-Grosjean and O. Buffet. Global multiprocessor real-time scheduling as a constraint satisfaction problem. In *Proceedings of the ICPP'09 Workshop on Real-time systems on multicore platforms : Theory and Practice (XRTS'09)*, 2009.

[2] A.M. Déplanche P.E. Hladik, H. Cambazard and N. Jussien. Solving a real-time allocation problem with constraint programming. *Journal of Systems and Software*, 81(1) :132–149, 2008.