Adaptation de l'algorithme JLA pour la génération de tâches temps-réel dans un modèle d'exécution Time-Triggered

Moez Rabai, Belgacem Ben Hedia
CEA LIST DRT
Point Courrier n 174
91191 Gif-sur-Yvette Cedex, France
Email : moez.rabai, belgacem.ben-hedia@cea.fr

Jean-Philippe Babau
Lab-STICC, UBO, UEB
20 avenue Le Gorgeu
29200 Brest, France
Email : jean-philippe.babau@univ-brest.fr

***Résumé*—La synthèse des applications temps-réel requiert l'allocation des traitements élémentaires, exprimés par un langage de description de haut niveau, sur des tâches temps-réel. A cette fin, on définit des stratégies et des algorithmes qui assistent le concepteur dans cette allocation en garantissant le respect des aspects fonctionnels et extra-fonctionnels de l'application. Dans cet article, on propose une adaptation de l'algorithme Joined Late Activation pour assurer l'allocation de traitements élémentaires sur un modèle de tâches "Time Trigerred". On propose ensuite une évaluation multi-critères (le nombre des changements de contexte, la latence moyenne, la flexibilité) de cet algorithme en comparaison avec deux approches extrêmes (une seule tâche pour l'application et une tâche par traitement élémentaire).**

I. Introduction

Les systèmes de contrôle/commande sont chargés de surveiller l'évolution des processus et de contrôler leur comportement. La mise au point des politiques de contrôle s'appuie sur des modèles de haut-niveau d'abstraction tels que les graphes de flot de données [8]. Pour passer de ce modèle de haut-niveau à un modèle d'exécution bas niveau, les développeurs doivent construire une structure de tâches temps-réel ; soit un problème classique d'allocation d'actions sur des tâches. Le modèle d'exécution bas niveau retenu pour la suite de cet article est un modèle basé sur le paradigme "Time Trigerred"(TT)[10], où le déclenchement des tâches est dirigé par le temps.

Le contexte des systèmes embarqués temps réel impliquent des ressources d'exécution limitées et le respect de contraintes de temps. Ce contexte nécessite l'optimisation des ressources d'exécution, soit un nombre limité de tâches pour réduire le coût de changements de contexte. De plus, la structure des tâches générée doit répondre aussi à des critères de qualité de service tel que la latence moyenne et la flexibilité laissée à l'OS temps-réel pour exécuter ces tâches.

Dans cet article, on propose une adaptation de l'algorithme de génération de la structure de tâches "Joined Late Activation" (JLA) [6] pour un modèle d'exécution TT et de le confronter à deux autres approches. Cette confrontation est basée sur trois critères : le nombre de changements de contexte, la latence moyenne de l'application et la flexibilité. Nous commençons par présenter l'état de l'art lié au sujet dans le chapitre II. Notre modèle fonctionnel de description haut niveau et le modèle d'exécution bas niveau sont présentés dans le chapitre III. On présente ensuite dans le chapitre IV, l'algorithme JLA ainsi que son adaptation. Nous appliquons cet algorithme à un cas représentatif des applications temps-réel ainsi qu'une évaluation multi-critères de cet algorithme dans le chapitre V. Enfin, nous concluons dans le chapitre VI.

II. État de l'art

Plusieurs études ont abordé le problème de l'allocation dans un contexte embarqué temps-réel. Pour réduire le nombre des tâches générées, la méthode DARTS [9] propose des politiques de regroupement, qui s'appuient sur les cohésions temporelles, séquentielles, les cohésions de contrôle et les cohésions fonctionnelles.

Dans l'article [13], les auteurs présentent une transformation automatique de SCADE [3] vers une application temps-réel multitâche. La mise en place de tâches s'appuie sur des composants fonctionnels.

Les auteurs de l'article [7] présentent un schéma complet pour la génération d'une application multitâche multipériodique à partir d'un langage de haut niveau d'abstraction nommé "PRELUDE". Leur approche génère un grand nombre des tâches qui n'est pas supporté par les RTOS actuels.

Wang and Shin [14] ont développé une approche visant la séparation des parties réutilisables. Cette approche, intéressante du point de vue ingénierie, ne prend pas en compte les aspects extra-fonctionnels tels que les paramètres temps-réel.

Plusieurs études comme [16] se sont intéressées aux problèmes du placement des tâches dans une architecture multiprocesseurs sans aborder la problématique de regroupement des traitements élémentaires dans les tâches .

L'outil commercial Simulink [2], largement utilisé, offre au concepteur un accompagnement pour la génération du code. Il offre la possibilité de générer une application multitâches où les fonctions activées avec la même fréquence sont regroupées dans la même tâche. Les priorités des tâches générées sont basées sur RMA ("Rate Monotonic Analysis") [11]. La

génération d'applications multitâches n'est possible que pour les applications où les fréquences des événements externes et internes sont harmoniques.

Dans [4], les auteurs étudient l'allocation des fonctions sur des tâches et le placement des tâches dans une architecture distribuée.

Dans ce travail, on s'intéresse à d'allocation des fonctions sur des tâches pour une architecture monoprocesseur en adaptant l'algorithme JLA pour un modèle d'exécution TT.

III. Modèle de conception haut niveau et modèle d'exécution bas niveau

A. Modèle de conception haut niveau

Il existe plusieurs outils et langages de haut niveau d'abstraction (UML MARTE [12] , SysML [15]...) pour la modélisation et la conception des applications temps-réel. Dans la suite de cet article, on considère que le comportement de toute application temps-réel embarquée peut être représenté à l'aide d'un graphe de flot de données synchrone avec des conditions d'activation. On appelle ce graphe : graphe de causalité. Ce formalisme intermédiaire permet de représenter toutes les informations nécessaires pour la génération de structure de tâches. Une transformation de modèle permet de passer de la description fournie par les outils et langages de haut niveau vers notre formalisme intermédiaire.

Le graphe de causalité est un graphe orienté $G\,(T, L, D, A)$ tel que :

1) $T\,(t_1, t_2, ..., t_n)$ représente l'ensemble des traitements élémentaires.
2) $L\,(l_1, l_2, ..., l_n) \subset (T \times D \times T)$ représente l'ensemble des arcs qui représentent les interactions entre les traitements élémentaires.
3) $D\,(d_1, d_2, ..., d_n)$ représente l'ensemble des données échangées par les traitements élémentaires par ces liens. Une donnée d_i peut être simple ou composée , dans notre modèle elle se présente sous la forme d'une seule entité : $d_i = \{d_{i1}, d_{i2}, ..., d_{in}\}$.
4) $A\,(a_1, a_2, ..., a_n)$ représente l'ensemble des conditions d'activation (qui peuvent être des horloges logiques) .

Un arc $l_i = (t_p, d_j, t_q)$ transmet la donnée d_j de la sortie de t_p à l'entrée de t_q. Sur le graphe, un arc l_i est tagué par la donnée d_j et il est représenté par $\xrightarrow{d_j}$.

Un traitement élémentaire t_i peut correspondre à la destination de plusieurs arcs l_i.

Si un arc l_i relie la sortie d'un traitement élémentaire t_i à l'entrée d'un autre traitement élémentaire t_j, alors t_i précède t_j et on note $t_i \prec t_j$. Pour chaque traitement élémentaire t_i, on associe une condition d'activation a_i. Alors un traitement élémentaire t_i ne peut être exécuté que si les deux conditions suivantes sont satisfaites :

1) Pour tout j tel que $t_j \prec t_i$, L'exécution de t_j est terminée.
2) La condition d'activation a_i est satisfaite.

B. Modèle d'exécution bas niveau

Le modèle d'exécution bas niveau de notre approche est un modèle cadencé par le temps TT. Les activités du système sont fonctions d'un cadencement temporel statiquement défini.

Pour chaque tâche τ_i on définit une horloge temps-réel h_i qui représente les dates où les entrées et les sorties de la tâche peuvent être observées. L'ensemble des horloges de toutes les tâches τ_i de l'application constitue son horloge H_{base}. L'horloge H_{base} définit toutes les dates où le système peut être observé. Dans notre approche toutes les horloges h_i sont des horloges régulieres dans le sens qu'il y a la même durée temporelle entre deux dates [5].

Pour une première approche, on considère que les tâches sont périodiques. La Figure 1 représente le modèle temporel d'une tâche temps-réel. La tâche τ possède une date de début au plus tôt $debut$ et une date de fin au plus tard fin. L'OS temps-réel OASIS [5] se charge d'exécuter les tâches constituant l'application selon l'algorithme d'ordonancement Earliest Deadline First [11].

Les communications entres les tâches sont mappées dans des variables partagées OASIS nommées variables temporelles [5]. Ces variables offrent la gestion de l'historique des valeurs d'une variable $(couple < valeur, temps >)$.

Si une tâche τ possède une variable temporelle V à exposer aux autres tâches, la valeur de la variable partagée entre les dates $debut$ et fin est sa valeur à la date $debut$: V(début).

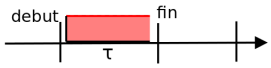

Figure 1: Modèle temporel d'une tâche temps-réel TT

C. Construction d'une structure de tâches

La conception d'une application temps-réel consiste à l'allocation des traitements élémentaires sur des tâches temps-réel $\Pi\,(\tau_1, \tau_2, ..., \tau_n)$. La fonction de l'allocation des traitements élémentaires t_i sur des tâches τ_j est : χ de T dans Π ,$t_i \rightarrow \tau_j$.

Pour garantir l'équivalence fonctionnelle entre le graphe de causalité G et l'ensemble des tâches générées, on doit montrer que les données sont produites et consommées par les tâches dans le même ordre que dans le graphe de causalité G. Pour chaque tâche, il y a au moins un point de synchronisation temporelle qui sépare sa fenêtre d'exécution de celles des tâches produisant les données exigées. Ainsi toutes les données exigées par une tâche sont disponibles quand nécessaire. Les relations de précédence entre les traitements élémentaires impliquent des relations de précédence entre les tâches produites. (Figure 2)

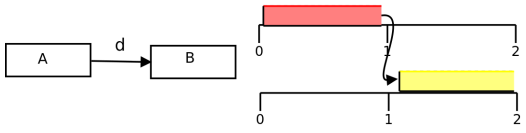

Figure 2: La relation de précédence entre deux tâche

La construction d'une structure multitâche à partir d'une description haut niveau nécessite la définition des algorithmes pour guider le concepteur.

Dans la suite de l'article, on étudie l'adaptation de l'algorithme JLA pour un modèle d'exécution TT.

IV. L'ALGORITHME JLA ET SON ADAPTATION POUR LE MODÈLE D'EXÉCUTION TT

Le modèle haut niveau présenté par les auteurs de l'article [6] est représenté par un graphe orienté acyclique qui a pour entrées des évènements externes e_i qui déclenchent son exécution. L'ensemble des blocs fonctionnels F_i et des arcs l_i représentent les fonctionnalités du système. Les arcs l_i reliant les blocs fonctionnels F_i sont purement des signaux d'activation et ne transportent pas de données. Des blocs o_i marquent la fin de l'exécution du flot de données.

Les blocs fonctionnels F_i sont équivalents aux traitements élémentaires t_i dans notre modélisation haut niveau.

Dans notre modèle haut niveau, on ne représentent pas les blocs o_i et la fin de l'exécution du flot de données est détectée par la restitution des données finales.

Dans notre approche, l'exécution des traitements élémentaires est déclenchée par les conditions d'activation a_i. Les arcs l_i reliant les traitements élémentaires t_i transportent des données d_i et ne sont pas purement des signaux d'activation.

Pour les auteurs de l'article [6], les pires temps d'exécution γ_i des blocs fonctionnels F_i sont connues et pour tout j, l'échéance ("deadline") Δ_i du chemin $P(e_i, o_j)$ est également donnée. Dans notre modélisation haut niveau, on ne dispose pas d'échéances minimales de chaque chemin et des pires temps d'exécution des traitements élémentaires.

L'algorithme de génération des tâches à partir des blocs fonctionnels F_i, développé par les auteurs de l'article [6], est basé sur les trois points suivants :

1) Une tâche τ est une chaîne de blocs fonctionnels et elle est entièrement incluse dans un chemin.
2) Un bloc fonctionnel, qui possède plusieurs entrées, est placé au début d'une tâche et est son point d'entrée unique.
3) Si un bloc fonctionnel possède plus d'une sortie, la tâche τ est prolongée le long du chemin P de plus courte échéance : La tâche τ ne peut pas contenir un bloc qui appartient à un chemin P' d'échéance Δ' inférieure à l'échéance Δ du chemin P, autrement elle activerait une tâche qui la préempterait immédiatement.

Les deux premiers points sont applicables à notre modélisation. Pour adapter le troisième point de l'algorithme à notre modèle haut niveau, la tâche générée est prolongée le long du plus court chemin. Cette adaptation vient du fait que dans un premier temps, les pires temps d'exécution des traitements élémentaires, non fourni par le modèle de haut niveau, sont considérés comme égaux.

V. ÉTUDE DU CAS ET ÉVALUATION

Afin d'illustrer notre étude, nous considérons une application composée de 8 traitements élémentaires définie par la FIGURE 3. Dans toute l'étude, l'application est supposée monopériodique, d'où la présence d'une seule condition d'activation a_1. la plate-forme matérielle cible est monoprocesseur.

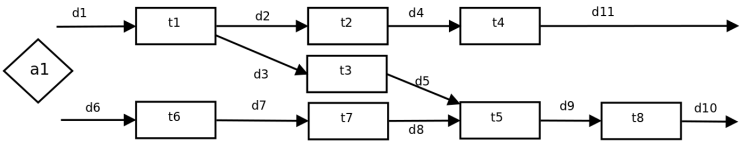

FIGURE 3: Graphe d'une application temps-réel simple

L'application de l'algorithme JLA adapté à notre approche fournit quatre tâches liées par les relations de précédence définies dans la FIGURE 4.

Tâche	Traitements élémentaires
τ_1	t1,t2,t4
τ_2	t3
τ_3	t5,t8
τ_4	t6,t7

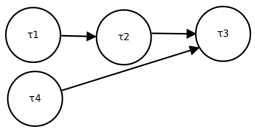

FIGURE 4: Relations de précédence entre les tâches générées

En prenant en compte les relations de précédence et la monopériode égale à huit, un cadencement possible est présenté sur la FIGURE 5.

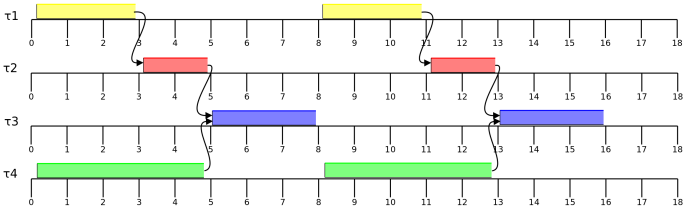

FIGURE 5: Cadencement des tâches générées par JLA

Pour évaluer l'algorithme JLA, on le compare à deux stratégies extrêmes : une seule tâche pour toute l'application et une tâche pour chaque traitement élémentaire.

Les critères d'évaluation sont :

– Le nombre de changements de contexte : transférer l'usage du processeur d'une tâche à une autre exige un changement de contexte qui implique des coûts supplémentaires.
– La latence moyenne de l'application : la latence de l'application correspond au délai de bout en bout entre le changement d'état de l'environnement physique et la restitution des résultats correspondants en sortie du système.
La latence moyenne de l'application sur plusieurs cycles : $L_{moyenne} = moyenne_k(f_{i,k} - s_{i,k})$, avec $f_{i,k}$ et $s_{i,k}$ respectivement la date de fin et la date de début du cycle k de l'application.
– La flexibilité du cadencement généré : soit deux traitements élémentaires t_1 et t_2 tels que $t_1 \prec t_2$, le regroupement de ces deux traitements en une seule tâche laisse plus de flexibilité pour OASIS pour leurs exécutions entre le début et la fin de la fenêtre temporelle allouée à la tâche. Ce qui permet de mieux exploiter les temps

processeurs non utilisés par un traitement pour répondre aux besoins en temps d'exécution d'un autre traitement dont l'estimation du temps d'exécution n'est pas correct.

En analysant le cadencement temporel de la FIGURE 5, on remarque qu'on peut mieux exploiter le temps processeur en décalant à gauche la deuxième itération de trois quantums (FIGURE 6). Ceci est possible grâce à la sémantique des variables temporelles OASIS.

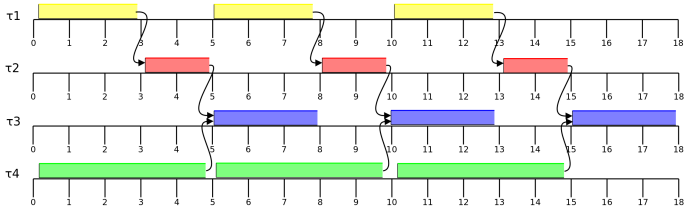

FIGURE 6: Cadencement optimisé des tâches générées par JLA

Pour évaluer La latence moyenne, on calcule le nombre des cycles qu'on peut exécuter pendant une mono-période. Pour évaluer la flexibilité, on a choisit pour métrique le nombre maximal des traitements élémentaires liés par des relations de précédence contenus dans une tâche τ. Pour l'algorithme JLA la flexibilté a pour valeur trois, puisque pour la tâche τ_1 on a trois traitements élémentaires liés par des relations de précédence.

Les métriques d'évaluation de la latence et de la flexibilité ne sont pas mises au point, la formalisation de ces métriques fera l'objet des prochaines études.

Pour déterminer le nombre de changements de contexte, on utilise le logiciel de simulation Cheddar [1].

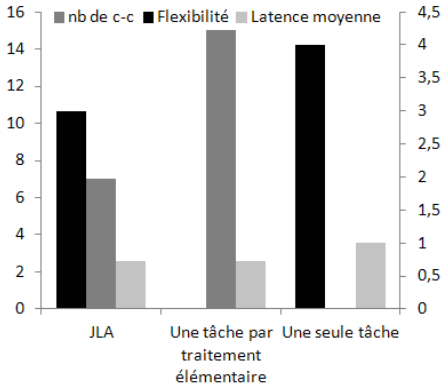

FIGURE 7: comparaison avec les deux approches extrêmes

La FIGURE 7 montre une comparaison multi-critères entre JLA et les approches extrêmes. Le nombre de changements de contexte (nb de c-c), la latence moyenne et la flexibilité sont présentés par de colonnes voisines pour chaque approche.

JLA présente l'avantage de préserver l'aspect concurrentiel de l'application et d'offrir la possibilité de "pipeliner" les tâches et ainsi réduire la latence moyenne de l'application par rapport à la stratégie d'une seule tâche pour tous les traitements élémentaires. Cet algorithme présente aussi l'avantage de garder une certaine flexibilité et de réduire le nombre de changements de contexte par rapport à l'approche d'une tâche par traitement élémentaire.

La FIGURE 8 montre le profil multi-critères de l'algorithme JLA.

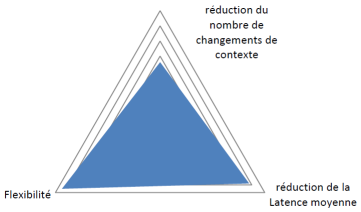

FIGURE 8: Évaluation multi-critères de l'algorithme JLA

VI. CONCLUSION ET PERSPECTIVES

Dans cet article, on présente l'adaptation de l'algorithme JLA pour l'allocation de traitements élémentaires au modèles d'exécution Time-Triggered. L'évaluation de cette adaptation montre l'intérêt d'un tel algorithme par rapport au regroupement de tous les traitements élémentaires dans une seule tâche pour préserver l'aspect concurrentiel de l'application et offrir la possibilité de "pipelinage". L'algorithme JLA permet de minimiser les coûts supplémentaires de changements de contexte et garde une bonne flexibilité par rapport au cas où on a un traitement élémentaire par tâche. Concernant les travaux futurs, on envisage de formaliser des métriques d'évaluation de la latence moyenne et de la flexibilité et de développer d'autres stratégies de regroupements des traitements élémentaires dans des tâches temps-réel.

RÉFÉRENCES

[1] [en ligne] http ://beru.univ-brest.fr/ singhoff/cheddar/index-fr.html.

[2] [en ligne] http ://www.mathworks.com.

[3] [en ligne]http ://www.esterel-technologies.com/products/scade-suite/.

[4] A.Mehiaoui, S.Tucci-Piergiovann, J.P.Babau, and L. Lemarchand. Optimizing the deployment of distributed real-time embedded applications. RTCSA '12, pages 400–403, Washington, DC, USA, 2012. IEEE Computer Society.

[5] C.Aussagues and V.David. A method and a technique to model and ensure timeliness in safety critical real-time systems. pages 2–12, ICECCS '98.

[6] C.Bartolini, G.Lipari, and M.Natale. From functional blocks to the synthesis of the architectural model in embedded real-time applications. RTAS '05, pages 458–467, Washington, DC, USA, 2005. IEEE Computer Society.

[7] C.Pagetti, J.Forget, F.Boniol, M.Cordovilla, and D.Lesens. Multi-task implementation of multi-periodic synchronous programs. *Discrete Event Dynamic Systems*, 21(3) :307–338, September 2011.

[8] Jack B. Dennis. Data flow supercomputers. *Computer*, 13(11) :48–56, 1980.

[9] H.Gomaa. *Software Design Methods for Concurrent and Real-Time Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1993.

[10] H. Kopetz and al. The design of large real-time systems : The time-triggered approach. In *IN PROC. 16TH REAL-TIME SYSTEMS SYMPOSIUM*, 1995.

[11] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, January 1973.

[12] UML MARTE. [en ligne]http ://www.omgmarte.org/.

[13] S.Bliudze, X.Fornari, and M.Jan. From model-based to real-time execution of safety-critical applications : Coupling SCADE with OASIS. In *Embedded Real Time Software and Systems*, ERTS2, February 2012.

[14] S.Wang and Kang G. Shin. An architecture for embedded software integration using reusable components. In *CASES*, pages 110–118, 2000.

[15] SysML. [en ligne]http ://www.sysml.org.

[16] W.Zheng, Q.Zhu, M.Natale, and A.Vincentelli. Definition of task allocation and priority assignment in hard real-time distributed systems. RTSS '07, pages 161–170, Washington, DC, USA, 2007. IEEE Computer Society.