Revue ARIMA

Ordonnancement de tâches hiérarchiques interdépendantes sous des exigences temporelles et objectif d'efficacité

Ismail Assayad

Equipe Architecture des Systèmes
ENSEM, Oasis Casa, Maroc
iassayad@gmail.com

RÉSUMÉ. L'introduction d'applications à haute performance comme les applications multimédia dans les systèmes embarqués a amené les constructeurs à offrir des plateformes embarquées capable d'offrir une puissance de calcul importante qui permet de répondre à l'évolution croissante des exigences futures de ces applications. L'une des solutions adoptées est l'utilisation de plateformes multiprocesseurs. Dans ce papier nous proposons une méthodologie *exploratoire* d'ordonnancement pour des logiciels modélisés sous forme de tâches hiérarchiques, collaboratives, interdépendantes et à échéances locales. L'ordonnancement répond aux (1) exigences *temporelles* locales des tâches et permet (2) une exploitation *efficace* de plateformes multiprocesseurs.

ABSTRACT. The introduction of high-performance applications such as multimedia applications into embedded systems led the manufacturers to offer embedded platforms able to offer an important computing power which makes it possible to answer the increasing requirements of future evolutions of these applications. One of the adopted solutions is the use of multiprocessor platforms. In this paper we propose an *exploratory* methodology of scheduling software modelled in the form of hierarchical, collaborative and interdependent hierarchical tasks with local deadlines. Scheduling answers (1) the local *temporal* requirements of tasks, and allows (2) an *effective* exploitation of multiprocessor platforms.

MOTS-CLÉS : Systèmes embarqués multiprocesseurs, Exploration, Parallélisme, Ordonnancement temps-réel.

KEYWORDS : Mutliprocessor embedded systems, Exploration, Parallelism, Real-time scheduling.

1. Introduction

Les applications embarquées de haute performance comme la compression vidéo, la transmission de paquets, HDTV, motivent l'utilisation de plateformes configurables, hétérogènes, et qui offrent plusieurs unités de traitement (eg. Wasabi/Cake, famille des réseaux de processeurs Intel IXP, etc.). Ces architectures offrent des avantages significatifs de prix, de performance et de flexibilité. Néanmoins, la complexité de ces systèmes embarqués multiprocesseurs rend la programmation et l'analyse du logiciel difficiles, ce qui mène à des performances inefficaces du logiciel et du matériel. Cette complexité est croissante et soulève le besoin d'une méthodologie de synthèse conjointe, permettant la description explicite de systèmes composés de logiciels et de matériels, et un ordonnancement du logiciel orienté par l'efficacité du système en terme de l'impact mutuel en performance du logiciel et du matériel.

La sémantique collaborative non préemptive est largement utilisée dans la spécification de logiciels. Ceci s'explique par le fait que les ordonnancements non préemptifs sont plus faciles à implanter, ont un surcoût prévisible et inférieur à celui des algorithmes préemptifs, et sont donc plus proche des modèles théoriques [10]. Par ailleurs l'utilisation de langages comme SystemC [1] dans la modélisation et la conception de systèmes industriels témoigne de cet intérêt.

Nous proposons une méthodologie de synthèse de logiciel composé de tâches hiérarchiques et interdépendantes qui *combine les exigences temps-réel locales des tâches avec un objectif d'efficacité du matériel*. La méthodologie se décompose en deux étapes. La première étape est la synthèse d'un ordonnanceur temps-réel au niveau logiciel qui définit un ordre partiel sur l'ensemble des tâches hiérarchiques. Cet ordonnanceur est indépendant du matériel mais prend en compte les interactions avec le matériel à travers les dépendances de données et de précédences. La deuxième étape est celle du placement. Cette étape définit l'ordonnanceur des tâches du logiciel au niveau matériel. L'efficacité globale de l'ordonnanceur logiciel et matériel peut alors être estimée par le concepteur après l'évaluation de son impact en performance sur le modèle de l'architecture matérielle.

L'ordonnanceur synthétisé permet d'éviter les cas de violation des exigences temporelles du logiciel ce qui simplifie l'exploration d'architectures du système pour le concepteur. De plus l'ensemble des points de contrôles de l'ordonnanceur du niveau logiciel est réduit à un ensemble de tâches hiérarchiques ce qui permet de diminuer sa complexité. D'autre part, l'ordonnanceur du niveau logiciel est indépendant du matériel et est donc généré une seule fois pour les différents cycles de conception. Par conséquent, l'impact de l'ordonnanceur sur le temps global de l'exploration s'en trouve réduit.

2. Etat de l'art

[10] présente une étude d'ordonnançabilité pour un système de tâches périodiques avec des dates de disponibilités arbitraires et sans préemption mais sans temps creux. Dans [14] une tâche est définie par une date de disponibilité, une échéance par rapport à cette date, une période et une durée d'exécution. Plusieurs travaux concernant l'ordonnancement de tâches sous des contraintes de distance dites bout-à-bout on été présentés [8, 11, 13]. [15] étudie l'ordonnançabilité d'un système de tâches périodiques en utilisant l'algorithme "rate monotonic" et en assurant une bonne utilisation des processeurs.

[9] donne des conditions d'ordonnançabilité qui permettent de déterminer si un système de tâches satisfait ses contraintes en utilisant l'algorithme EDF qui reste non optimal dans le cas multiprocesseur. Il y a d'autres critères d'optimalité qui intéressent le monde temps-réel comme par exemple optimiser le nombre de processeurs [12], minimiser le temps de réponse des tâches [16], ou les temps de communication [2, 17]. La plupart des résultats obtenus dans ces travaux concernent des cas particuliers comme par exemple le cas des périodes et des durées d'exécutions divisibles entre elles [12], et des algorithmes optimaux pour des problèmes particuliers comme la multiplication de matrices [2].

Dans ce papier nous proposons une méthodologie *exploratoire* contrairement aux algorithmes présentant une unique solution. Elle s'inscrit dans le cadre de l'exploration d'architecture logicielle et matérielle avec un impératif de *combiner les exigences temps-réel du logiciel avec un objectif d'efficacité du matériel*. Elle permet de synthétiser un ordonnanceur pour des tâches *hiérarchiques*, *interdépendantes avec des temps creux d'attente, et des échéances locales*. Cette méthodologie est utilisée dans P-WARE [5, 6, 7], un framework d'exploration d'architectures logicielles et matérielles.

3. Modèle du système

3.1. Modèle matériel

Une requête de transaction TR est un tuple $\langle type, input\ values, output\ values \rangle$, où $input\ values$ désigne les valeurs des paramètres d'entrées nécessaires pour l'exécution de la transaction T correspondante à la requête par le composant matériel destination, $output\ values$ dénote les valeurs de sorties.

Les instructions composant une tâche logicielle produisent des requêtes de transactions à destination des composants matériels. L'exécution des transactions correspondantes à ces requêtes par le matériel est définie au niveau du modèle du comportement des composants matériels.

Un composant matériel C est constitué d'un comportement et d'une interface $\langle I_i, O_j \rangle$. Le comportement est constitué d'un arbitre de requêtes, un contrôleur de transaction, la spécification temporelle des opérations des transactions, et le transacteur qui génère les requêtes de sortie d'un composant sur un de ses ports de sortie. I_i est l'ensemble des ports d'entrée des requêtes TR de C. O_j est l'ensemble des ports de sortie des requêtes TR.

Une connexion est un tuple $\langle O, I_1, \ldots I_n, type \rangle$ où O est un port de sortie des requêtes TR du composant source ; $I_1, \ldots I_n$ sont les ports d'entrée des requêtes TR transmises à partir du port O ; $type$ est le type de la connection, Il est "bloquant" pour les envois et les appels de fonctions bloquants et "non bloquant" pour les interruptions et les notifications.

Une architecture matérielle est un couple $\langle \mathcal{C}, \mathcal{B} \rangle$ où

– $\mathcal{C}$ est l'ensemble des composants.

– $\mathcal{B}$ est l'ensemble des connections.

La sémantique complète des composants matériels peut être trouvée dans [5, 6].

3.2. Modèle logiciel

Un logiciel est constitué d'un ensemble de tâches hiérarchiques ou simples.

Une tâche simple t est un tuple $\langle S, dep, \delta \rangle$ où S est l'ensemble des requêtes de transactions TR associées aux instructions composant la tâche, dep est l'ensemble des dépendances sources pour S, δ est le temps d'exécution pire cas de t.

Une architecture $\mathcal{A}$ d'une tâche t est un tuple $\langle C, \mathcal{D} \rangle$ où C est l'ensemble des contraintes d'ordonnancement sur les temps de début de t, $\mathcal{D}$ est la fonction de placement sur S vers les composants matériels. Dans ce papier nous considèrons que toutes les requêtes d'une tâche simple sont placées sur un même composant [1].

Une tâche hiérarchique t est un tuple $\langle op, t1 \ldots tn, dep, D, C \rangle$ où op est un opérateur de combinaison valant soit seq, par ou or pour la composition séquentielle, parallèle, ou non déterministe, respectivement, en plus des opérateurs de boucles, D est l'échéance sur le temps d'exécution des sous-tâches $t1 \ldots tn$, dep est l'ensemble des dépendances sources pour t, C est l'ensemble des contraintes de placement sur les temps de début des sous-tâches $t1, \ldots tn$.

Les contraintes de placement pour une tâche hiérarchique t définissent l'ordonnancement de ses sous-tâches $t1, \ldots tn$ sur le matériel. Le WCET d'une tâche peut être considéré inconnu, ou peut être dépendant du placement. Ce dernier cas est traité comme si l'opérateur or avait été utilisé pour spécifier toutes les différentes valeurs du WCET.

L'architecture d'une tâche hiérarchique t est égale à l'union des architectures de ses sous-tâches $\mathcal{A}_{ti}$ augmentée des contraintes de placement C des sous-tâches $t1, \ldots tn$: $\cup_{i=1}^{i=n} \mathcal{A}_{ti} \cup C$.

4. Synthèse de contraintes

Nous étendons ici la technique de synthèse de contraintes présentée dans [4] à des tâches hiérarchiques, indéterministes et sujettes à des échéances locales. Cette technique vise à synthétiser un ordonnanceur au niveau logiciel, indépendamment du matériel. Cet ordonnanceur consiste en un ensemble de contraintes [2] sur les temps de débuts des tâches.

4.1. Description

Granularite La technique de synthèse est compositionnelle, ce qui signifie que les contraintes synthétisées pour le logiciel sont l'union des contraintes synthétisées pour chacune des tâches parmi une partition des tâches du logiciel. En choisissant une petite partition, la complexité de la synthèse peut être diminuée. Une partition est un ensemble de tâches hiérarchiques dont les sous-tâches ont les mêmes contraintes. A titre d'exemple si les sous-tâches $(t1, \ldots tn)$ d'une tâche hiérarchique $(T = op\ t1, \ldots tn)$ ne sont impliquées dans aucune dépendance avec une autre tâche, alors le temps de début des sous-tâches $(t1, \ldots tn)$ dépend uniquement du temps de début de T. Par conséquent, les sous-tâches $(t1, \ldots tn)$ peuvent ne pas faire partie de la partition sur laquelle la synthèse au niveau logiciel est opérée, en les mettant dans un même élément dans la partition. Cet élément peut être T ou une tâche contenante vérifiant la même condition.

1. Le cas général se ramène facilement au précédent.
2. Ces contraintes définissent un ordre partiel alors que l'ordre total est défini par la phase de placement.

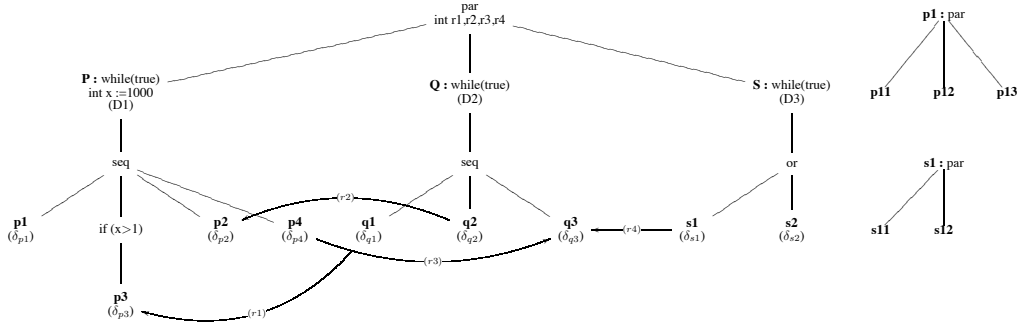

Figure 1. *Trois tâches hiérarchiques P, Q et S et les deux sous-tâches p1 et s1*

L'exemple de la figure 1 contient 9 tâches simples. La partition maximale $\mathcal{P}$ pour cet ensemble de tâches est composée de 9 tâches, $\mathcal{P} = \{p_{11}, p_{12}, p_{13}, p_3, p_2, p_4, q_1, q_2, q_3\}$, dont les tâches p_{11}, p_{12}, p_{13} et s_{11}, s_{12} ont les mêmes contraintes. Les dépendances entre les sous-tâches de $p1$ ne sont pas présentées dans cette figure. Ci-après, une tâche se réfère à un élément d'une telle partition du logiciel.

Exigences Une tâche possède une échéance locale comme exigence sur son temps de fin. Par exemple, l'échéance d'une tâche $while$ est l'échéance de chaque itération de la boucle à partir du temps de début de cette itération.

Sortie La technique produit des contraintes sur les temps de début des tâches, ainsi que des contraintes sur les temps d'exécution des tâches dont le temps d'exécution est inconnu. La solution est celle d'un programme linéaire entier, ILP.

Dans la présentation ci-après, nous faisons l'hypothèse que les données ne sont pas échangées entre les tâches par envoi de messages, mais des communications à travers la mémoire partagée sont utilisées. Notons néanmoins que le cas d'envoi de messages se réduit au premier en traitant les communications comme des tâches.

4.2. Extension

La technique présentée dans [4] calcule des contraintes pour des boucles concurrentes et pouvant inclure des branches conditionnelles dans leurs itérations. Nous étendons cette technique aux tâches hiérarchiques présentées auparavant, qui peuvent également impliquer des choix de tâches sous forme $or\ t1, \ldots, tn$. Les points-clés de cette extension sont les suivants :

– Pour chaque tâche, *propager* les exigences et les exprimer comme des contraintes sur les temps d'attente des tâches et les variables de temps d'exécutions inconnus. Le résultat de cette propagation est dénoté par $\Psi(w,\delta)$.

– Pour chaque tâche, calculer les contraintes sur les temps de début, qui vont garantir les contraintes $\Psi(w,\delta)$. Pour cela, le temps d'attente d'une tâche est *exprimé* par une relation entre les temps de début de cette tâche et les tâches sur lesquelles elle dépend. Cette relation est dénotée $\Phi(b, w,\delta)$. Ensuite, en utilisant les contraintes sur ce temps d'attente données par $\Psi(w,\delta)$, et la relation $\Phi(b, w,\delta)$, des contraintes sont *inférées* sur les variables b et δ. Ces contraintes sont dénotées $\Delta(b,\delta)$.

Les contraintes $\Delta(b,\delta)$ sont consistantes par construction. En effet, Φ caractérise l'ensemble des interactions consistantes avec les temps d'attentes objets des contraintes Ψ.

Par conséquent, cette technique consiste à calculer $\Psi(w,\delta)$ et $\Phi(b, w,\delta)$, et génère

des contraintes de la forme :

– $\bigwedge\ \Delta(b_{ti}, \delta_j)$,

– $\bigvee_{C_k}\ (\bigwedge\ \Delta(b_{ti}, \delta_j)\ \bigwedge\{b_t = \perp, t \in C_k\})$ dans le cas de tâches avec des choix d'exécution.

L'ensemble de variables δ_j désigne l'ensemble des variables de temps d'exécution inconnus, et chaque ensemble C_k est un ensemble de choix pour lequel les contraintes correspondantes $\bigwedge\ \Delta(b_{ti}, \delta_j)\ \bigwedge\{b_t = \perp, t \in C_k\}$ sont valides.

L'ordonnanceur du logiciel est alors soit la solution d'un programme linéaire entier, ILP, ou un ensemble de paires de programmes linéaires entiers avec l'ensemble des choix correspondants, $\langle ILP_k, C_k\rangle$.

4.3. Complexité

La complexité de la technique est dominée par les facteurs suivants :

1) Le nombre quadratique des tâches, qui est la complexité des opérations de calcul de $\Phi(w,\delta)$ et l'inférence de $\Delta(b,\delta)$.

2) La somme cumulée des choix d'exécutions sur l'ensemble des tâches Or.

3) Un exposant du nombre des dépendances inter-tâches. En effet la relation Φ est potentiellement non linéaire. Des temps creux dus aux temps d'attente pour synchronisation des tâches peuvent intervenir dans le calcul de la relation Φ. Ceci dépend des temps de fin et temps de début d'une ou plusieurs tâches sources et leur tâche destination, respectivement.

Par conséquent, cette technique génère un nombre de contraintes ILP au plus égal à $\sum_{i=1}^{I} \Pi_{k,k\geq 1} n_d(n_s+1)^{k_i}$ et sa complexité au pire cas vaut $\bigcirc(N^2 \times I \times \Pi_{k,k\geq 1} n_d(n_s+1)^k)$ où :

– N est le nombre de tâches de la partition du logiciel,

– I est le nombre de tâches hiérarchiques indéterministes multiplié par le nombre de leurs choix d'exécution,

– k est le nombre maximal de dépendances ou d'hyper-dépendances dont le nombre de sources vaut n_s et le nombre de destinations vaut n_d, sur tous les choix d'exécutions possibles ($k = max_{i=1}^{I} k_i$).

Notons néanmoins que les applications embarquées temps-réel visées par cette méthodologie telles que les applications multimédia contiennent un nombre de dépendances inter-tâches très petit par rapport aux dépendances entre sous-tâches, et relativement constant par rapport aux différentes versions d'une même application (eg. standards d'encodage vidéo [3]). Cette remarque réduit l'impact de l'exposant sur la complexité de la technique dans la conception de ces applications.

4.4. Exemple

Considérons l'exemple de la figure 1 avec les données d'échéances $D1 = 10, D2 = 10$, et $D3 = 5$; et les WCET suivants : $\delta_{p_2} = 1, \delta_{p_3} = 1, \delta_{p_4} = 1, \delta_{q_1} = 1, \delta_{q_2} = 4, \delta_{q_3} = 1, \delta_{s_1} = 1, \delta_{s_2} = 1$; et en considérant que le WCET de p_1 est inconnu.

Nous obtenons alors l'ensemble des contraintes C ci-après où $x = b_{p_1} - b_{q_1}$ et $y = b_{s_1} - b_{q_1}$ sont les distances temporelles entre les tâches p_1 q_1 et s_1.

Quand s2 est exécuté, $b_{s_1} = \perp$ et

$$\begin{cases} (1) & \delta_{p_1} >= 5 + x & \wedge & \delta_{p_1} <= 6 & & \\ (2) & \delta_{p_1} >= 4 + x & \wedge & \delta_{p_1} <= 5 & \wedge & x \leq 2 \\ (3) & \delta_{p_1} >= 1 + x & \wedge & \delta_{p_1} <= 4 & \wedge & x \leq 2 \\ (4) & \delta_{p_1} >= 0 & \wedge & \delta_{p_1} <= 1 & & \end{cases}$$

Quand s1 est exécutée, $b_{s_2} = \perp$ et

$$\begin{cases} (5) & \delta_{p_1} >= x - y - 2 \wedge \delta_{p_1} >= x + 5 \wedge \delta_{p_1} <= x + 6 \wedge \delta_{p_1} <= 7 \\ (6) & \delta_{p_1} >= x - y - 2 \wedge \delta_{p_1} >= x + 4 \wedge \delta_{p_1} <= x + 5 \wedge \delta_{p_1} <= 7 \wedge x <= 2 \\ & \wedge y >= -7 \\ (7) & \delta_{p_1} >= x - y - 2 \wedge \delta_{p_1} >= x + 4 \wedge \delta_{p_1} <= x + 5 \wedge \delta_{p_1} <= 7 \wedge x <= 2 \\ & \wedge y >= -8 \wedge y <= -6 \\ (8) & \delta_{p_1} >= x - y - 2 \wedge \delta_{p_1} >= x + 4 \wedge \delta_{p_1} <= x + 5 \wedge \delta_{p_1} <= 7 \wedge x <= 2 \\ & \wedge y >= -7 \wedge y <= -4 \\ (9) & \delta_{p_1} >= x - y - 2 \wedge \delta_{p_1} <= x + 5 \wedge \delta_{p_1} <= 7 \wedge y >= -8 \wedge y <= -4 \\ (10) & \delta_{p_1} >= x - y - 2 \wedge \delta_{p_1} >= x + 4 \wedge \delta_{p_1} <= x + 5 \wedge \delta_{p_1} <= 7 \wedge x <= 2 \\ & \wedge y >= -8 \wedge y <= -6 \\ (11) & \delta_{p_1} >= x + 1 \wedge \delta_{p_1} <= x + 4 \wedge x <= 2 \wedge y >= -6 \\ (12) & \delta_{p_1} >= x + 1 \wedge \delta_{p_1} <= x + 4 \wedge x <= 2 \wedge y >= -7 \wedge y <= -6 \\ (13) & \delta_{p_1} >= x + 1 \wedge \delta_{p_1} <= x + 4 \wedge x <= 2 \wedge y >= -8 \wedge y <= -7 \\ (14) & \delta_{p_1} <= x + 1 \wedge \delta_{p_1} <= 5 \wedge x <= 2 \wedge y >= -6 \\ (15) & \delta_{p_1} >= x + 1 \wedge \delta_{p_1} <= 5 \wedge x <= 2 \wedge y >= -7 \wedge y <= -6 \\ (16) & \delta_{p_1} >= x + 1 \wedge \delta_{p_1} <= 5 \wedge x <= 2 \wedge y >= -8 \wedge y <= -7 \end{cases}$$

Figure 2. *Les ensembles de contraintes pour l'exemple de la figure 1*

A titre d'exemple, quand s_2 est exécutée ($b_{s_1} = \perp$), et quand l'ordonnancement de q_1 et p_1 vérifie $x = b^{q_1} - b^{p_1} \leq 2$ ce qui signifie que la distance temporelle entre les tâches hiérarchiques Q et P est égale ou inférieure à 2, alors la contrainte sur le temps d'exécution de p_1 est : $\delta^{p_1} = \max(\delta^{p_{11}}, \delta^{p_{12}}, \delta^{p_{13}}) \in [0, 6]$

L'ensemble des contraintes synthétisées dans la section 4 constituent un ensemble de systèmes de contraintes linéaires S sur les temps de débuts et les temps d'exécution de certaines tâches.

Pour calculer l'ordonnanceur du niveau matériel, nous définissons tout d'abord la fonction de placement des tâches. En effet, il faut ajouter au système de contraintes indépendant du matériel des contraintes de placement indiquant comment sont placées les tâches sur les ressources (composants) qui doivent les exécuter. Ensuite, soit ces systèmes restent inchangés dans le cas où les tâches sont placées sur différents (groupes de) processeurs car ce placement n'introduit pas de nouvelles contraintes, soit ils sont augmentés par conjonction avec les contraintes de placement, soit ils sont recalculés (voir exemple ci-après). Puis, les solutions finales sont obtenues en prenant les temps de début au plus tôt des tâches, ce qui correspond à la fonction objectif $min \sum_i b_{t_i}$.

A titre d'exemple, le placement sur 3 processeurs un pour chacune des tâches P, Q et S de l'exemple de la figure 1, ne produit pas de contraintes supplémentaires aux tâches. Lorsque 2 processeurs uniquement sont utilisés, 1 pour P et Q et 1 pour S, l'ordonnanceur peut être défini par une des deux approches suivantes :

– Recherche exhaustive, i.e. recalcul des systèmes de contraintes après recalcul de la contrainte Ψ en rajoutant aux contraintes de la relation de consistance Φ les deux contraintes de placement suivantes :

$$b_{p_1} + \delta_{p_1} + \delta_{p_2} + \delta_{p_3} + w_{p_3} + w_{p_2} > b_{q_1} \bigvee b_{q_1} + \delta_{q_2} + \delta_{q_3} + w_{q_3} > b_{p_1}$$

– Recherche partielle, i.e., garder l'ensemble des systèmes de contraintes de niveau logiciel inchangés et rajouter une sur-approximation des contraintes de placement en utilisant les échéances. Les nouveaux systèmes restent consistants. Par conséquent l'ordonnaceur du niveau matériel est :

$$\left\{ \begin{array}{c} b_{p_1} + D1 > b_{q_1} \\ S \\ min\ \sum_i b_{q_i} + \sum_j b_{p_j} + \sum_k b_{s_k} \end{array} \right. \vee \left\{ \begin{array}{c} b_{q_1} + D2 > b_{p_1} \\ S \\ min\ \sum_i b_{q_i} + \sum_j b_{p_j} + \sum_k b_{s_k} \end{array} \right.$$

4.5. Efficacité

Après la définition d'une fonction de placement et le calcul de l'ordonnanceur final, les performances du système sont estimées afin d'en évaluer l'efficacité. Pour simplicité, nous présentons ici cette évaluation pour une sous-tâche hiérarchique p_1. Les sous-tâches de p_1 sont présentées dans le listing 1.

```
p1: while(i>0)
    p11: x=add(x0,read(&M[i]));
    p12: dec(i); write(&M[i],x);
    p13: write(&z,mult(x,read(&c)));
```

Listing 1 – Les sous-tâches de p1

L'architecture matérielle possède les propriétés suivantes :

TR	read	write	add	mult	dec
Latences des T	2	2	2	2	1
T conflictuelles	-	-	mult	add	-
T parallèles	write, dec	read, dec	read, write	-	read, write

Les latences de l'écriture, la lecture, l'addition et la multiplication valent 2 cycles. Contrairement aux deux premières transactions, l'addition et la multiplication doivent être exécutées sur différents cycles. [6, 5] détaillent la modélisation d'architectures matérielles.

Comme les sous-tâches de p_1 ne font pas partie de la partition de départ, leur ordonnanceur n'est pas synthétisé. Pour de telles tâches, la méthodologie est compositionnelle et peut donc être réappliquée comme auparavant pour les sous-tâches en prenant la contrainte sur p_1 comme échéance. Dans ce cas, il faut prendre en compte les temps d'attentes dus aux dépendances cycliques entre tâches, càd, celles dues aux tâches de lecture de la variable i et sa décrémentation d'une part, et la lecture/écriture de M d'autre part.

P-WARE est utilisé pour évaluer les performances de l'ordonnancement global du système (Débit, bande passante, temps d'exécution, etc.). Un ordonnancement séquentiel possède un temps d'exécution de 6 cycles et donne un débit d'une sortie tous les 6 cycles. Alors que l'ordonnancement pipeline sur les deux processeurs P1 et P2 produit un débit meilleur pour le même temps d'exécution.

Ordonnancement	Placement P1	Placement P2	δ_{p_1}	Débit
Séquentiel	p11, p12, p13	-	6	< 0.17
Pipeline	p11, p12	p13	6	0.25

5. Application d'encodage vidéo

5.1. Description

Afin de montrer l'intérêt et l'applicabilité de notre approche, nous présentons ici les résultats de calcul d'ordonnanceurs par synthèse de contraintes pour la conception d'une application industrielle : implémentation efficace d'une plateforme sur puce pour encodage vidéo MPEG-4 à tranches.

Les tâches de cet encodeur sont présentées dans les figures 3 et 4.

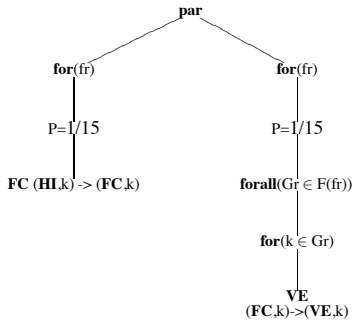

Figure 3. *Deux sous-tâches : simple FC et composée VE de l'encodeur*

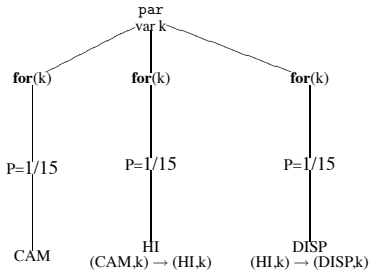

Figure 4. *Trois tâches simples de période P en interaction avec l'encodeur*

La plateforme sur puce d'encodage vidéo fonctionne de la manière suivante : tout d'abord les images de la caméra CAM sont envoyées à l'interface utilisateur HI avec le format du caméra, i.e., RGB. Ensuite l'interface utilisateur réalise certaines opérations, comme l'agrandissement d'image, avant sa transmission à l'affichage DISP (figure 4). En parallèle à l'affichage, les autres tâches, le convertisseur de format FC et l'encodeur vidéo VE, tournent sur les processeurs. Ils convertissent puis encodent les images 640x480. FC est en charge de la transformation des images vers le format de VE, i.e., YUV, avant de

les transmettre à VE pour qu'elles soient encodées dans le format H264 sur le matériel Cake présenté dans la figure 5.

L'objectif de cette application est d'utiliser la méthodologie pour trouver une implémentation qui satisfasse les exigences de l'application avec une optimisation de la consommation de la bande passante sur la plateforme Cake. Pour cela le type de parallélisation du logiciel peut être au niveau donnée ou tâche et le nombre de processeurs peut varier dans chacun des système sur puce interconnectés de Cake (figure 5).

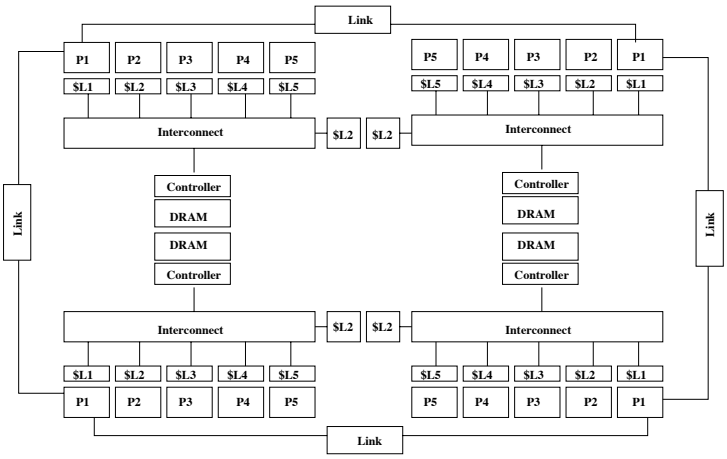

Figure 5. *Cake avec une configuration à quatre systèmes*

5.2. Synthèse de contraintes logicielles et matérielles

Regardons ici les résultats de l'application de notre méthodologie à la conception de la plateforme. La synthèse prend en compte les exigences de l'application, i.e., les périodes d'activation des tâches, ainsi que les interactions des tâches matérielles et logicielles.

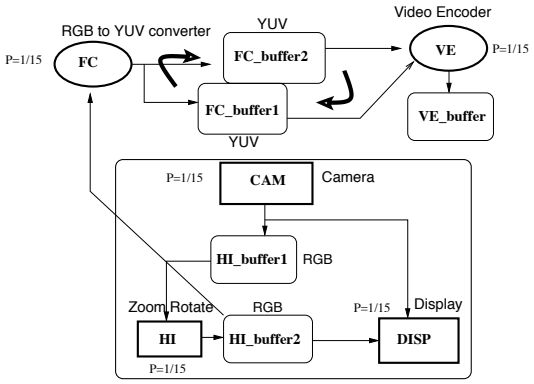

Figure 6. *Interactions du logiciel et du matériel environnant*

Ces interactions sont constituées de dépendances de données stockées dans des tampons mémoires selon le schéma indiqué dans la figure 6. Les exigences temporelles consistent en les périodes des boucles $(1/15)$ pour les tâches du logiciel FC, et VE ; et la moitié de période $(1/30)$ pour les tâches matérielles CAM, HI et DISP.

La procédure de synthèse de contraintes fournit un ordonnanceur (figure 7) et une contrainte sur le temps d'exécution du logiciel (Equation (5)). Dans ces figures, le temps de début et les temps d'exécution d'une tâche X sont indiqués par les notations b_X et δ_X, respectivement.

$$\begin{cases} (1) \quad b_{\text{HI}} = b_{\text{CAM}} + \frac{1}{30} & \text{HI activé} \frac{1}{30} \text{ après CAM} \\ (2) \quad b_{\text{DISP}} = b_{\text{HI}} + \frac{1}{30} & \text{DISP activé } \frac{1}{30} \text{ après HI} \\ (3) \quad b_{\text{FC}} = b_{\text{HI}} + \frac{1}{30} & \text{FC activé } \frac{1}{30} \text{ après HI} \quad \text{(ordonnanceur logiciel)} \\ (4) \quad b_{\text{VE}} = b_{\text{FC}} + \frac{1}{30} & \text{VE activé } \frac{1}{30} \text{ après FC} \quad \text{(ordonnanceur logiciel)} \end{cases}$$

Figure 7. *Contraintes synthétisées de l'ordonnanceur*

A titre d'exemple, dans cet ordonnanceur, la première contrainte stipule que l'interface doit commencer son exécution après le début de l'exécution de l'acquisition de la caméra par un trentième de secondes.

D'autre part, le synthèse dérive également une contrainte sur les temps d'exécution de FC et VE considérées comme paramètres dans le modèle :

$$(5) \quad \delta_{\text{FC}} + \delta_{\text{VE}} \leq \tfrac{1}{15} \qquad \text{Contraintes sur les temps d'exéc de VE et FC à satisfaire}$$

Cette contrainte fixe la condition sous laquelle l'ordonnanceur respecte les exigences du modèle. L'équation (5) est la contrainte de temps d'exécution.

Considérons que le temps d'exécution de FC est :

$$\delta_{FC} \leq \frac{2}{3} \times \frac{1}{25}$$

Il s'en suit que l'échéance sur le temps d'exécution de l'encodeur est :

$$\delta_{VE} \leq \frac{1}{25}$$

La figure 8 montre un ordonnancement compatible avec l'ensemble des contraintes.

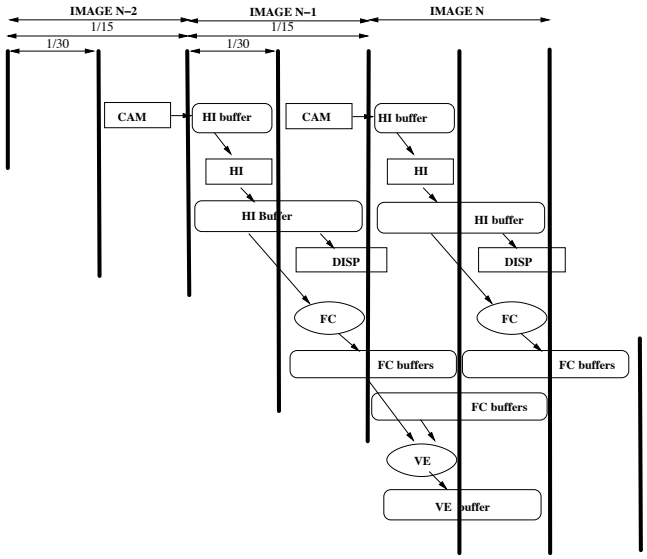

Figure 8. *Ordonnancement faisable de l'application*

VE est parallélisé sur les processeurs disponibles dans les SoCs Cake alors que FC tourne en séquentiel sur un des processeurs, et selon le modèle du logiciel FC et VE tournent en exclusion mutuelle. Par conséquent l'ordonnanceur au niveau matériel n'introduit pas de nouvelles contraintes sur VE ou FC. Pour les placements suivants, il vaut (3), (4) avec la contrainte (5).

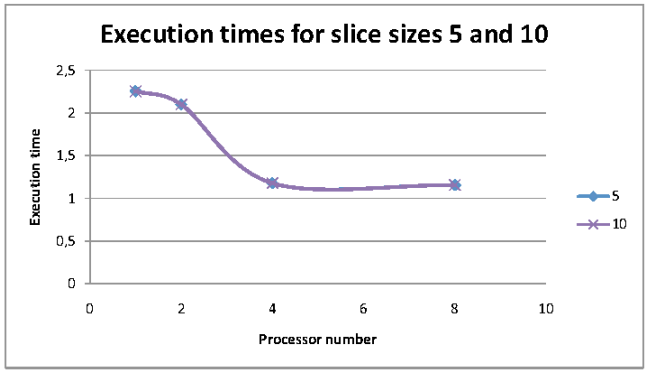

Figure 9. *Temps d'exécution pour des placements parallèles par tâches*

Les figures 9, 10 et 11 montrent les résultats de performances sur des trames de 25 images. Ces résultats proviennent de différents placements de la tâche hiérarchique VE. Les placements parallèles par tâches consistent à placer une tâche sur le même processeur et à placer des (groupes de) tâches différents sur des processeurs différents. La figure 9

montre que ces placements violent la contrainte $\delta_{VE} \leq \frac{1}{25}$ dans le cas des tranches de tailles 5 et 10 macroblocs. Les tailles supérieures ont des performances similaires.

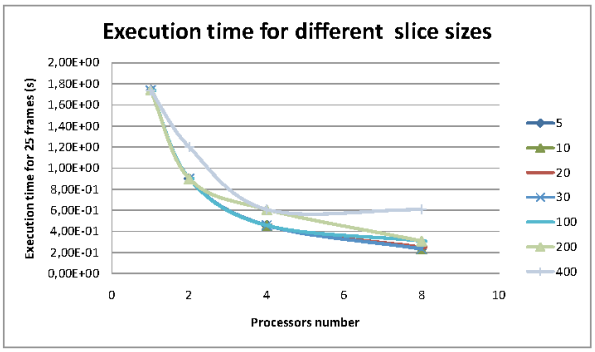

Figure 10. *Temps d'exécution pour des placements parallèles par tranches de données*

Les placements parallèles par tranches de données consistent à placer des unités indépendantes composées d'une ou plusieurs tranches sur des processeurs différents. Ces placements donnent des résultats meilleurs que les précédents. En effet, ils permettent de satisfaire la contrainte de l'ordonnanceur à partir de 2 processeurs sauf pour des tranches de tailles égales (ou supérieures) à 400 macroblocs. Comme le montre la figure 10, le gain en temps d'exécution au delà de 4 processeurs n'est pas très significatif. Il ressort de cela que, compte tenu des bandes passantes consommées et des débits réalisés par les différents placements de la figure 11, le placement par tranches de 100 macroblocs sur 4 processeurs est l'implémentation qui satisfait les exigences de l'encodeur et qui soit la plus efficace en terme de bande passante matérielle et de débit de sortie.

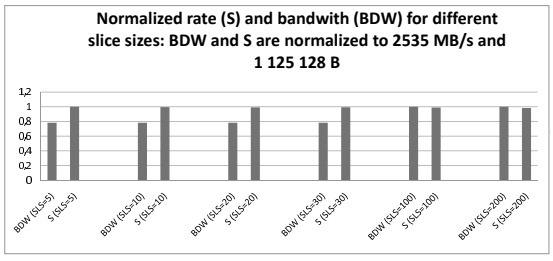

Figure 11. *Bande passante matérielle (BDW) et débit (S) pour des placements parallèles par tranches de données*

6. Conclusion et perspectives

Nous avons proposé une méthodologie exploratoire pour la synthèse d'ordonnanceurs pour logiciels embarqués modélisés par des tâches hiérarchiques interdépendantes et sujet à des échéances temps-réel locales. Cette méthodologie permet tout d'abord de calculer

un ensemble de systèmes de contraintes linéaires définissant les ordonnanceurs au niveau logiciel. Ensuite, différents placements des tâches et leurs sous-tâches sur l'architecture matérielle définissent des ordonnanceurs au niveau matériel du logiciel. Enfin, l'évaluation de performances matérielles et logicielles de ces placements permet de comparer l'efficacité des différents ordonnancements.

Plusieurs facteurs permettent de réduire la complexité de l'ordonnanceur. Premièrement, le concepteur peut contrôler cette complexité de calcul de l'ordonnanceur par le choix d'une partition des tâches du logiciel. Deuxièmement l'ordonnanceur logiciel est indépendant du matériel et peut n'être généré qu'une seule fois.

Cette méthodologie a été utilisée dans le cadre du framework P-WARE et a été appliquée à la conception d'une plateforme industrielle d'encodage utilisant une version à tranches de l'encodeur vidéo MPEG-4. Nous avons en particulier montré qu'une configuration utilisant un seul système Cake avec uniquement quatre processeurs et un encodage à tranches de cent macroblocs permet un encodage temps-réel efficace.

Un effort de développement spécifique est nécessaire pour automatiser la génération de l'ordonnanceur à partir de descriptions textuelles des modèles des tâches logicielles et des composants matériels. Cette synthèse automatique du système permettra d'offrir un gain intéressant en temps de cycles d'exploration et en temps d'implantation d'une part, et d'autre part, d'autres réalisations d'applications industrielles pourront aisément servir pour valider la méthodologie proposée.

Remerciements. L'auteur tient vivement à remercier les relecteurs anonymes pour leur commentaires et suggestions.

Références

[1] SystemC. http ://www.systemc.org.

[2] A. Aggarwal, A. K. Chandra, and M. Snir. On communication latency in pram computations. In *SPAA'89*, pages 11–21, New York, NY, USA, 1989. ACM.

[3] I. Assayad, Ph. Gerner, S. Yovine, and V. Bertin. Modelling, analysis and implementation of an on-line video encoder. In *DFMA'05. IEEE Computer Society*, 2005.

[4] I. Assayad and S. Yovine. Compositional constraints generation for concurrent real time loops with interdependent iterations. In *I2CS'05*. Springer Verlag, LNCS, 2005.

[5] I. Assayad and S. Yovine. System platform simulation model applied to multiprocessor video encoding. *IEEE symposium on industrial embedded systems*, 2006.

[6] I. Assayad and S. Yovine. P-WARE : A precise and scalable component-based simulation tool for embedded multiprocessor industrial applications. *EUROMICRO DSD*, 2007.

[7] Ismail Assayad and Sergio Yovine. Modelling and exploration environment for application specific multiprocessor systems. In *HASE '07*, pages 433–434, Washington, DC, USA, 2007. IEEE CS.

[8] Richard Gerber, William Pugh, and Manas Saksena. Parametric dispatching of hard real-time tasks. *IEEE Trans. Computers*, 44(3) :471–479, 1995.

[9] Kees Goossens, John Dielissen, Om Prakash Gangwal, Santiago González Pestana, Andrei Rădulescu, and Edwin Rijpkema. A design flow for application-specific

networks on chip with guaranteed performance to accelerate SOC design and verification. In *DATE'05*, pages 1182–1187, Washington, DC, USA, 2005. IEEE CS.

[10] K. Jeffay, D. F. Stanat, and C. U. Martel. On non-preemptive scheduling of periodic and sporadic tasks. pages 129–139, 1991.

[11] Dong-In Kang, Richard Gerber, and Manas Saksena. Performance-based design of distributed real-time systems. In *RTAS '97*, page 2, Washington, DC, USA, 1997. IEEE CS.

[12] Jan H. M. Korst, Emile H. L. Aarts, and Jan Karel Lenstra. Scheduling periodic tasks with slack. *INFORMS Journal on Computing*, 9(4) :351–362, 1997.

[13] William Lindsay and Parameswaran Ramanathan. Dbp-m : A tech. for meeting end-to-end firm{m}{k} guarantee requirements in point-to-point net. In *LCN*, page 294, 1997.

[14] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1) :46–61, 1973.

[15] Dong-Ik Oh and Theodore P. Bakker. Utilization bounds for n-processor rate monotone scheduling with static processor assignment. *RTS*, 15(2) :183–192, 1998.

[16] John P. Penny, Paul J. Ashton, and A. L. Wilkinson. Data recording and monitoring for analysis of system response times. *Comput. J.*, 29(5) :396–403, 1986.

[17] S. Saez, J. Vila, and A. Crespo. Using exact feasibility tests for allocating real-time tasks in multiprocessor systems. *ecrts*, 00 :53, 1998.