Ordonnancement en ligne et temps réel de tâches matérielles multi-versions sur FPGA

Guy WASSI[1], Geoff LAWDAY[1], Amine BENKHELIFA[2], François VERDIER[2]

[1]School of Computing and Advanced Technologies, Bucks New University
Queen Alexandra Road, High Wycombe HP11 2JZ, UK

[2]Laboratoire ETIS, UMR 8051, ENSEA, Université de Cergy, France
6, avenue du Ponceau 95014 Cergy Pontoise Cedex France

guy.wassi@bucks.ac.uk, geoff.lawday@bucks.ac.uk
benkhelifa@ensea.fr, verdier@ensea.fr

Résumé – Ce papier s'inscrit dans la problématique de conception des RSoC (Reconfigurable System-On-a-Chip) disposant d'un système d'exploitation temps réel (RTOS). Nous étudions l'ordonnancement et le placement en ligne des tâches matérielles *multi-versions* sur FPGA partiellement reconfigurable. En effet, une tâche matériellle peut être synthetisée en plus d'une version et avoir plusieurs tailles et/ou formes avec les caractéristiques temporelles conséquentes. Nous évaluons l'apport de cette approche à tâches *multi-versions* sur quelques algorithmes d'ordonnancement dans un contexte en ligne. A travers des métriques pertinentes (taux de réjection des tâches, taux d'occupation, makespan), les résultats montrent que cette approche améliore significativement l'ordonnancement sans modifier sa complexité algorithmique. Par exemple, pour les tâches de classe de laxité B ayant 2 versions, le taux de rejection des tâches est diminué de 46% pour un temps d'exécution moyen de l'algorithme d'ordonnancement inchangé en comparaison à l'algorithme EDF (Earliest Deadline First) utilisant des tâches *mono-version*.

Abstract – The context of this paper is the design of reconfigurable System-On-a-Chip (RSoC) featuring a Real-Time Operating System (RTOS). We modeled and evaluated different algorithms implementing two services of such an RTOS : the online scheduling and placement of hardware tasks into the reconfigurable part (eg. FPGAs) of the RSoC. In this paper, we assess the so-called *multi-versions* tasks approach in which a hardware task might have one or many shapes and/or sizes. The results show that by making such efforts at design level the scheduling is improved without significantly increasing the algorithm complexity and execution time. For example, by using 2 versions per tasks of laxity class B, the tasks rejection ratio decreases by 46% without changing the scheduling algorithm execution time compared to EDF (Earliest Deadline First) scheduling using single version tasks.

1 Introduction et travaux similaires

Aujourd'hui, la possibilité de reconfiguration partielle et dynamique des FPGAs permet d'effectuer un multiplexage spatial et temporel des circuits logiques (tâches) sur leur surface (fig. 1). Ceci fait des FPGAs une plateforme adaptée à l'implémentation des applications embarquées multi-tâches, dynamiques et fortement parallèles. Les FPGAs offrent de ce fait probablement le meilleur compromis architectural entre les solutions généralistes et flexibles à base de processeurs et les systèmes performants et dédiés à base d'ASICs. Mais l'ordonnancement ciblant un FPGA est beaucoup plus complexe que l'ordonnancement mono ou multi-processeurs. En effet, il s'approche d'un ordonnancement multi-processeurs hétérogènes à nombre de processeurs continuellement variable dans le temps. Par ailleurs, on distingue l'ordonnancement hors ligne et l'ordonnancement en ligne. Dans l'ordonnancement hors ligne, le flot de l'application est connu d'avance et l'ordonnancement optimal de l'application peut être calculé avant son démarrage. Par contre, dans un contexte en ligne [5], le temps d'arrivée des tâches ainsi que leur temps d'exécution sont non prédictibles ; l'ordonnancement est alors ré-effectué en ligne en fonction des évènements survenus (arrivée ou fin d'une tâche, etc...).

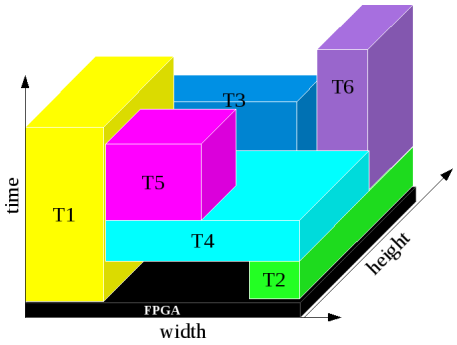

FIG. 1 – Ordonnancement des tâches sur FPGA (vue 3D).

De nombreux auteurs [6][3][4][5] ont effectué des travaux sur l'ordonnancement et le placement temps réel pour FPGAs. Dans [6] sont proposés des algorithmes d'ordonnancement 1D et 2D en ligne et temps réel des tâches sporadiques alors que [3] se focalise sur l'ordonnancement temps réel, préemptif mais

hors-ligne et 1D des tâches périodiques. Comparé à [3] qui utilise aussi des tâches multi-versions, nous traitons des tâches sporadiques dans un contexte en ligne, 2D et non-préemptif. Nous utilisons un placement 2D commun à tous les algorithmes d'ordonnancement simulés ici ; ceci permet une comparaison de ces algorithmes à placeur équivalent. Ce placement est peu détaillé dans ce papier, car n'étant pas le but premier de notre étude. Dans la suite de ce papier nous présentons la modélisation du problème et les métriques pertinentes, puis nous présentons les algorithmes simulés et enfin, nous situons le contexte des simulations réalisées et commentons les résultats obtenus.

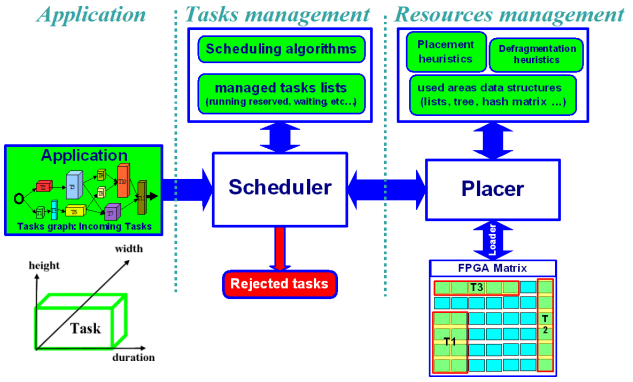

FIG. 2 – Modélisation du point de vue fonctionnel.

2 Modèles et métriques de performance

2.1 Modélisation du problème

Une représentation fonctionnelle du modèle de simulation est présentée figure 2.

- Le *FPGA* est une matrice 2D de ressources homogènes (fig. 1 et 2). $FPGA_{(W,H)}$ désigne un FPGA de largeur W, de hauteur H, et donc de taille $W \cdot H$, où $W \cdot H$ est le nombre blocs logiques indépendamment reconfigurables (eg. CLBs) et dynamiquement attribués aux tâches.
- Une *tâche matérielle* T_i est un bistream pré-routé et relogéable sur le FPGA. Elle est caractérisée par ses paramètres :
  - géométriques (largeur w_i , hauteur h_i, voir fig. 2) ; le *facteur de forme* de la tâche est $aspect\_ratio_i = \frac{w_i}{h_i}$.
  - temporels (temps d'exécution e_i, temps d'arrivée a_i, échéance d_i, fig. 2) ; on en déduit la *laxité* de la tâche
  $$l_i = d_i - a_i - e_i$$

  La quantité totale de ressources nécessaire à l'exécution d'une tâche $T_i[w_i, h_i, e_i, a_i, d_i]$ est exprimée par l'équation 1 où A_i est la taille de la tâche :
  $$Tres_i = w_i \cdot h_i \cdot e_i = A_i \cdot e_i \qquad (1)$$

  Dans un contexte temps réel, T_i doit respecter son échéance.
- Une *Application App* est composée de k tâches matérielles T_1, T_2,...T_k pouvant être représentées dans un graphe de tâches (fig. 2). De l'équation (1) on déduit la quantité de ressources nécessaire à l'exécution de $App[T_1, T_2, ...T_k]$:
  $$App_{res} = \sum_{i=1}^{k} w_i \cdot h_i \cdot e_i = \sum_{i=1}^{k} Tres_i \qquad (2)$$
- L'*Ordonnanceur* (Scheduler, fig. 2) met en oeuvre la politique d'ordonnancement des tâches de l'application. Il prend en compte les tâches au fur et à mesure de leur arrivée dans le système. En effet, dans un scénario en ligne, tous les paramètres de la tâche (temps d'arrivée compris) sont inconnus avant son arrivée. L'ordonnanceur détermine si possible une position (x_i,y_i) et une date de démarrage s_i pour chaque tâche T_i de l'application de telle sorte que T_i ne chevauche jamais à la fois en position et en temps avec aucune autre tâche T_j (fig. 1). Pour cela, il sollicite le *Placeur*. Les tâches ne pouvant respecter leur échéance sont rejetées (fig. 2).
- Le *Placeur* (resources manager, fig. 2) répond aux requêtes de placement de l'ordonnanceur. Il gère les ressources à l'aide d'une structure de données représentant l'état du FPGA. Dans notre cas, cette structure est un arbre binaire de recherche (Fig. 3) tel que décrit dans [1], auquel est associée une matrice de hachage [7] stockant et indexant rapidement les espaces libres sur le FPGA. Ces espaces libres constituent les feuilles de l'arbre. L'arbre et la matrice sont mis à jour à chaque insertion ou suppression de tâche. La complexité algorithmique de la recherche d'un noeud dans l'arbre est de $O(n)$ au pire cas, n étant le nombre de tâches présentes sur le FPGA.

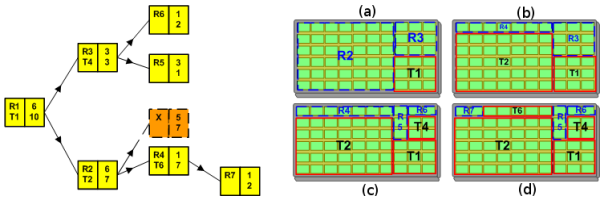

FIG. 3 – Utilisation d'un arbre pour représenter le FPGA.

Un exemple est décrit figure 3 où les tâches T_1, T_2, T_4 et T_6 sont successivement placées sur un $FPGA_{(10,6)}$. Sur la figure 3 (b) puis (c), le placement de la tâche T_4 dans le rectangle R_3 génère deux rectangles fils disjoints R_5 et R_6 qui sont inserrés dans l'arbre. A la fin de la tâche T_4, si R_5 et R_6 sont libres, le rectangle R_3 d'origine est reconstitué par suppression de ses rectangles fils (fig.3 (b)).

2.2 Métriques

Les métriques nous renseignent sur les performances de l'ordonnancement et du placement. On distingue essentiellement :

- ***Le facteur de charge d'une application :***
  Soit une application à k tâches $App[T_1, T_2, ...T_k]$ devant respecter son échéance D_{App} ; son facteur de charge ou taux d'utilisation des ressources sur un $FPGA_{(W,H)}$ est

déduit de l'équation (2) et vaut :

$$App_{load} = \frac{\sum_{i=1}^{k} w_i \cdot h_i \cdot e_i}{W \cdot H \cdot D_{App}} = \frac{App_{res}}{W \cdot H \cdot D_{App}} \quad (3)$$

– ***Ordonnançabilité :***
La condition nécessaire mais pas suffisante d'ordonnançabilité de *App* sur un $FPGA_{(W,H)}$ est :

$$App_{load} \leq 1 \quad (4)$$

– ***Le taux d'occupation moyen d'un*** $FPGA_{(W,H)}$ ***:*** Pour un $FPGA_{(W,H)}$ le taux d'occupation est donné par l'équation (5) où *makespan* est la durée d'ordonnancement ou durée réelle de l'application, et où le dénominateur représente la quantité totale de ressources disponible sur le $FPGA_{(W,H)}$ pendant la durée de l'application.

$$Occup\_ratio_{av} = \frac{App_{res}}{W \cdot H \cdot makespan} \quad (5)$$

– ***Le taux de réjection moyen des tâches :***
Pour une application à *k* tâches $App[T_1, T_2, ...T_k]$, le taux de réjection est donné par l'équation (6) où N_{reject} est le nombre de tâches n'ayant pas pu être placées.

$$Reject\_ratio_{av} = \frac{N_{reject}}{k} \quad (6)$$

– ***Le temps moyen d'exécution de l'algorithme d'ordonnancement*** *(fig. 7)* : Ce temps exprime le temps moyen pris par l'algorithme d'ordonnancement (et de placement sous-jacent) pour s'exécuter. Il est donné par l'expression ci-dessous (eq. 7) où *n* est le nombre d'invocation de l'ordonnanceur et $Exec\_Time(i)$ le temps d'exécution de l'algorithme d'ordonnancement à son *i-ème* appel.

$$Sched\_exec\_time_{av} = \frac{\sum_{i=1}^{n} Exec\_Time(i)}{n} \quad (7)$$

3 L'algorithme multi-versions

Cet algorithme d'ordonnancement convient aux applications ayant des tâches déclinées en plus d'une version (fig. 4). En utilisant une implémentation sur FPGA suivant l'arithmétique distribuée [2] par exemple, chaque tâche peut avoir une ou plusieurs formes et/ou tailles, avec les temps d'exécution correspondants (fig. 4). Une implémentation totalement parallèle utiliserait plus de ressources mais offrirait un débit de données plus élevé.

La formule (8) donne une expression très simplifée du rapport entre les tailles et les temps d'exécution de deux versions T_{i_normal} et T_{i_serial} d'une tâche T_i (fig. 4) :

$$\frac{A_{i\_normal}}{A_{i\_serial}} = \frac{w_{i\_normal} \cdot h_{i\_normal}}{w_{i\_serial} \cdot h_{i\_serial}} = \frac{e_{i\_serial}}{e_{i\_normal}} \quad (8)$$

où le produit $A_{i\_j} = w_{i\_j} \cdot h_{i\_j}$ est la taille de la version j de la tâche T_i (resp. e_{i_j} est le temps d'exécution de la version j de la tâche T_i).

En se basant sur l'équation (8) nous avons évalué l'algorithme multi-versions dans 3 différents cas de figures (fig. 4) :

– *Shape(1)* dans lequel chaque tâche a une version normale et une 2^{eme} version de facteur de forme inférieur à 1 (standing task), de taille moitié et de temps d'exécution double.
– *Shape(2)* qui est du *Shape(1)* auquel on ajoute une 3^{eme} version de taille et de temps d'exécution inchangés mais de facteur de forme moitié.
– *Shape(4)* qui est du *Shape(2)* auquel on ajoute une 4^{eme} version de facteur de forme supérieur à 1 (laying task), de taille moitié et de temps d'exécution double.

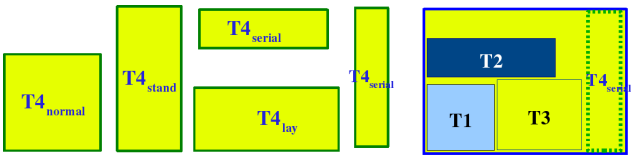

FIG. 4 – Tâches multi-versions (la tâche T4 a 5 versions)

L'algorithme multi-versions effectue un ordonnancement par liste dans lequel les tâches sont triées par ordre d'arrivée et sont maintenues dans la liste des tâches prêtes tant qu'elles peuvent encore respecter leur échéance. En cas de terminaison d'une tâche, la tâche en tête de liste est prioritaire et ainsi de suite.
En cas d'arrivée d'une nouvelle tâche, si l'état du FPGA n'a pas changé depuis la dernière tentative de placement des tâches de la liste, alors la nouvelle tâche est prioritaire pour le placement. En cas d'échec, elle est rajoutée dans la liste suivant sa priorité (ordre d'arrivée). L'algorithme choisit une version de la tâche élue parmi les versions existantes dans l'ordre de priorité suivant : *version normale*, puis *versions taille identique*, puis *versions taille réduite* (mais à temps d'exécution plus long) si et seulement si l'échéance de la tâche le permet.

Dans ses 3 déclinaisons Shape(1), Shape(2) et Shape(4), l'algorithme multi-versions est comparé à 3 autres algorithmes d'ordonnancement par liste fonctionnant sur le même principe, mais utilisant des tâches à une seule version :
– EDF (Earliest Deadline First scheduling),
– BSF (Biggest Size First) et SSF (Smallest Size First)
dans lesquelles l'échéance (resp. la taille) des tâches est le critère de priorité dans la liste triée des tâches prêtes.

Tous ces algorithmes sont de complexité $O(n)$ où *n* représente le nombre d'éléments dans la liste des tâches prêtes.

4 Résultats et Conclusion

Nous avons implémenté les différents algorithmes en C++ et réalisé les simulations sur un ordinateur avec processeur Intel Premium dual-core T2330, 1.6 GHz, 1MB de cache L2. Nous avons utilisé 50 jeux de 50 tâches et les résultats sont donc moyennés sur 50 valeurs. Les autres paramètres sont uniformément distribués dans les intervalles indiqués ci-après :
– Taille du FPGA : $largeur = 96$ et $hauteur = 64$.
– Tâches : Taille $\in [50, 1000]$ CLBs ;
facteur de forme ou aspect ratio $\in [\frac{1}{5}, 5]$,
laxité de classe A $\in [1, 10]$, B $\in [11, 50]$, C $\in [51, 100]$.

– Temps d'exécution $\in [5, 100]$ unités de temps.
– Facteur de charge des applications $App_{load} = 50\%$, pour la laxité de classe A.

Les figures 5, 7 et 6 montrent les résultats des simulations des algorithmes mono-versions (EDF, SSF BSF) et multi-versions *Shape(i)*, et ce pour différentes classes de laxité :

– Dans le cas de la *laxité de classe A*, l'algorithme multi-versions n'améliore de facon sensible la qualité de l'ordonnancement que si on a 4 versions par tâche (Shape(4)).
– Dans le cas de la *laxité de classe B*, utiliser deux versions par tâches (Shape(1)) permet de réduire de 46% (EDF) à 50% (SSF) le taux de réjection des tâches (fig. 5), d'augmenter de 18% (EDF) à 22% (SSF) le taux d'occupation (fig. 6) pour un temps d'exécution de l'algorithme augmenté de 10% comparé à l'algorithme SSF, et inchangé comparé à EDF et BSF (fig. 7).
– Dans le cas de la *laxité de classe C*, les résultats sont fortement améliorés dès l'utilisation de deux versions par tâches (Shape(1)). Le taux de réjection est quasiment nul ($\approx 1.5\%$, fig. 5) pour un taux d'occupation (fig. 6) amélioré de 31% (SSF) à 40% (EDF). Sachant que le *makespan* (fig. 8) devient significatif à taux de rejection nul, on remarque qu'il ne dépasse pas 180 unités de temps pour l'algorithme multi-versions alors qu'il depasse 195 pour les algorithmes EDF, BSF et SSF ayant pourtant un taux de rejection d'en moyenne 15% (fig. 5).

Nous avons à travers ce travail montré l'intérêt des algorithmes utilisant plusieurs versions de tâches matérielles. L'atout majeur de cette approche multi-versions est d'améliorer la qualité de l'ordonnancement et du placement sans augmenter de façon significative le temps d'exécution des algorithmes. L'étape suivant de cette étude sera d'une part d'analyser une éventuelle corrélation entre le partitionnement du FPGA et le facteur de forme des versions des tâches, et d'autre part d'évaluer le coût de cette approche en terme d'espace mémoire de stockage des tâches multi-versions.

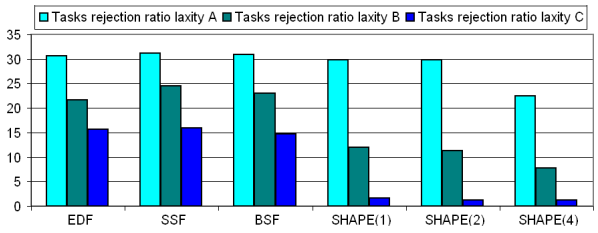

FIG. 5 – Taux de réjection des tâches (%)

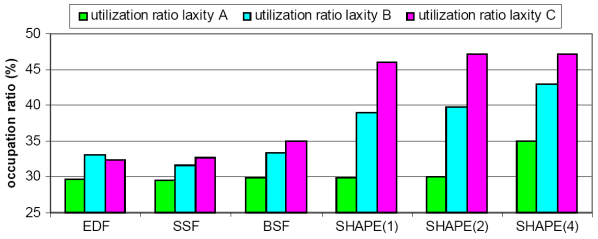

FIG. 6 – Taux d'occupation du FPGA(%)

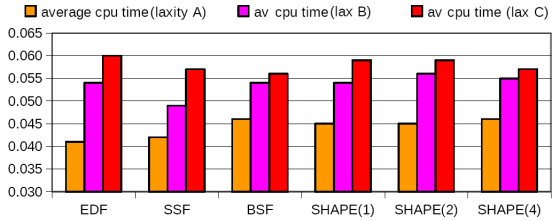

FIG. 7 – Temps d'exécution de l'algorithme (secondes)

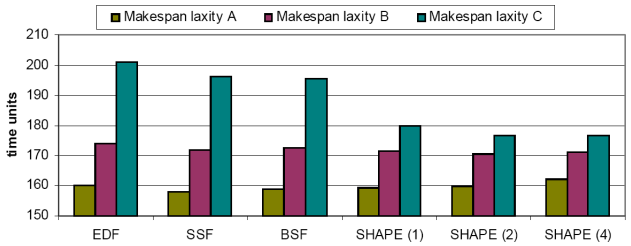

FIG. 8 – Makespan ou durée d'ordonnancement

Références

[1] Kastner R. Bazargan, K. and M. Sarrafzadeh. Fast template placement for reconfigurable computing systems. *IEEE Des. Test*, 17(1) :68–83, 2000.

[2] C. Bobda, A. Ahmadinia, J. Teich, and K. Danne. Generation of Distributed Arithmetic Designs for Reconfigurable Applications. *ARCS Workshops*, pages 205–214, 2004.

[3] K. Danne and M. Platzner. A heuristic approach to schedule periodic real-time tasks on reconfigurable hardware. In *International Conference on Field Programmable Logic and Applications*, pages 24–26, 2005.

[4] K. Danne and M. Platzner. An EDF schedulability test for periodic tasks on reconfigurable hardware devices. In *Proceedings of the 2006 ACM SIGPLAN/SIGBED conference on Language, compilers, and tool support for embedded systems*, pages 93–102. ACM New York, NY, USA, 2006.

[5] T. Marconi, Y. Lu, K.L.M. Bertels, and G. N. Gaydadjiev. Online hardware task scheduling and placement algorithm on partially reconfigurable devices. In *Proceedings of International Workshop on Applied Reconf. Computing (ARC)*, pages 306–311, March 2008.

[6] C. Steiger, H. Walder, M. Platzner, and L. Thiele. Online scheduling and placement of real-time tasks to partially reconfigurable devices. In *Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE*, pages 224–225, 2003.

[7] H. Walder, C. Steiger, and M. Platzner. Fast online task placement on FPGAs : free space partitioning and 2D-hashing. *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, page 8, 2003.