Gestion orienté qualité d'expérience de systèmes embarqués reconfigurables par réutilisation de modules

Alexis DUHAMEL[1,2], Sébastien PILLEMENT[1], Wiem KOUKI[2]

[1]Nantes Université, CNRS, IETR UMR 6164, F-44000 Nantes, France

[2]Capgemini Engineering, R&I Department, Rennes, France

{alexis.duhamel, wiem.kouki}@capgemini.com, sebastien.pillement@univ-nantes.fr

Résumé – La reconfiguration dynamique partielle des FPGA permet l'exploitation d'accélérateurs matériels échangeables en ligne. Dans le but de rendre ces systèmes les plus efficaces possibles, le partitionnement et l'ordonnancement d'applications à moindre coût sur ces ressources de calcul sont cruciaux. Dans cet article, nous proposons une méthode permettant de réutiliser des modules reconfigurables déjà instanciés lors d'amélioration ou de dégradation de paramètres qualités d'une application. Les résultats montrent que l'approche est capable de réutiliser les modules déjà implémenté pour réduire la latence entre modes d'exécution, tout en maintenant un haut niveau de qualité d'expérience.

Abstract – Dynamic partial reconfiguration of FPGA enables hot-swapping of hardware accelerators. Mapping and scheduling applications with little overhead is critical to making such systems as efficient as possible. In this paper, we introduce a method to reuse instanciated reconfigurable modules when upgrading or degrading an application's quality parameter. Results show our approach is capable of reusing already implemented modules to reduce latency between execution modes, while keeping a high level of quality of experience.

1 Introduction

Les plate-formes à base de FPGA reconfigurables dynamiquement ont des performances similaires aux processeurs graphiques GPU [1]. Leur gestion vise à permettre l'exécution d'applications type flot de données pour en améliorer les performances. L'objectif ici est de maximiser la qualité d'expérience d'une application ciblée en restant flexible face aux contraintes d'utilisation de ces ressources matérielles de calcul.

La gestion des ressources peut bénéficier de la réutilisation de modules déjà implémentés dans des régions reconfigurables [2]. Cette méthode réduit effectivement le nombre d'actions à prendre entre deux implémentations, mais à ce jour aucune approche ne cherche à l'exploiter durablement à l'exécution.

Nous proposons ici une méthode permettant d'exploiter cette technique dans le cadre de changements de paramètres d'application impactant la qualité d'expérience. Cette méthode fait usage d'un modèle de qualité d'expérience basé sur l'identification de modes d'exécution et de notre heuristique Q-Greedy présenté dans cet article pour maximiser le niveau de qualité pour des applications de traitement d'image.

2 Modèle de plate-forme

Les plate-formes cibles sont des systèmes embarqués SoPC à base de FPGA et de processeurs. L'architecture FPGA est conçue pour exploiter la reconfiguration dynamique partielle du FPGA, permettant le chargement à chaud de modules reconfigurables. Cette architecture, illustrée Figure 1 est représentative de la littérature [3].

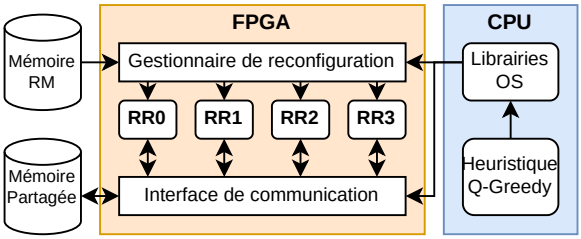

FIGURE 1 – Architecture considérée sur SoC Zynq 7000.

Cette architecture est composée d'un gestionnaire de reconfiguration dynamique, d'une interface de communication point-à-point, et de régions reconfigurables dotées de leur interface de communication.

Les régions reconfigurables (RR) contiennent des modules reconfigurables (RM). Ces derniers sont pré-générés sur une machine hôte et stockés dans une mémoire pour une utilisation sur cible. Les RRs sont composées d'un nombre différents d'éléments logiques. Cette hypothèse de RRs hétérogènes permet de pallier les problématiques de gaspillage de ressources en proposant des RRs plus grosses pour des tâches consommant plus de ressources, et inversement. Ceci impacte notamment les latences de reconfiguration qui dépendent de la taille des RRs.

Un processeur permet l'exécution de librairies de système d'exploitation telles que FOS [4] pour contrôler les RRs. Ces

librairies permettent de s'assurer que les RMs ont été correctement instanciés et que les données ont été correctement transmises à travers l'interface de communication.

Pour instancier un nouveau RM sur une RR, elle doit être reconfigurée préalablement via le port ICAP (*Internal Configuration Access Port*) par un gestionnaire de reconfiguration dynamique. Ce dernier est implémenté sur la partie statique du FPGA et récupère les bitstreams des RMs en mémoire à la demande du processeur avant de reconfigurer la RR. Ce processus prend un temps non-négligeable. Pour les tailles de RR considérées (un huitième jusqu'à un quart d'un FPGA Zynq 7000), la durée de ce processus est typiquement de l'ordre de 0.6 à 3ms pour des bandes passantes d'environ 400Mo/s [5].

3 Modèle d'application

Les applications sont représentées par leur graphe orienté acyclique (DAG). Ces graphes sont composées de noeuds représentant des tâches de l'application, et de liens entre les noeuds représentant des transferts de données. En particulier, nous nous intéressons au modèle de calcul *Scenario-Aware Data Flow* (SADF) [6]. Ce modèle permet la définition d'une application comportant différents modes d'exécution, ou scénarios. Le découpage en tâches de l'application peut être fait selon divers objectifs. Dans le cadre de nos travaux et du concept d'abstraction du matériel, nous considérons un découpage fonctionnel de nos applications pour ne pas introduire de contraintes supplémentaires dans la conception d'applications. Ce découpage permet aussi de simplifier l'introduction de différents modes d'exécution pour une application donnée.

Nous utiliserons comme exemple une implémentation FPGA d'un encodeur H264 agrémenté de tâches optionnelles de filtrage et de chiffrement afin d'illustrer notre approche. Le graphe de tâche de cette application est illustré Figure 2.

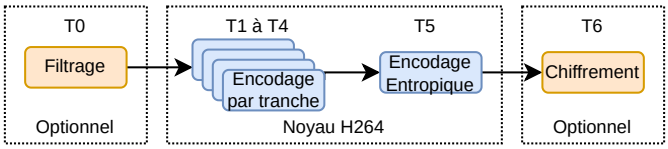

FIGURE 2 – Graphe de tâches de notre encodeur H264. Le noyau H264 est composé des tâches T1 à T5 et permet l'encodage de 4 tranches d'image en parallèle. Optionnellement, l'image peut être filtrée avant encodage avec la tâche T0, et l'image encodée peut être chiffrée avec la tâche T6.

4 Modèle de qualité d'expérience

La qualité d'expérience est une métrique subjective traduisant la satisfaction d'un utilisateur. Le modèle proposé a pour but de donner un score quantifiable et objectif d'un mode d'exécution d'une application donnée vis-à-vis de ses paramètres.

4.1 Mode d'exécution

Les modes d'exécution d'une application sont définis par leur combinaison de paramètres (eg : résolution d'image, fréquence en images par seconde de l'encodeur...). Autrement dit, chaque changement sur l'un de ces paramètres introduit un changement fonctionnel de l'application. Des paramètres peuvent être liés aux tâches optionnelles. Par exemple, notre encodeur H264 possède deux modes optionnels : un permettant le filtrage par détection de contours (Sobel), réduction de bruit (filtre médian) ou non de l'image, et un autre permettant le chiffrement AES ou non de l'image encodée. Le nombre maximum de modes d'exécution est borné par les combinaisons de paramètres, car certaines combinaisons de paramètres pour une application donnée peuvent ne pas être implémentable. Ceci peut être une conséquence des limites techniques de la plate-forme qui ne peut exécuter un mode d'exécution avec tous les paramètres aux valeurs les plus élevées, ou un choix de conception en amont.

Ainsi, chaque mode d'exécution est défini par les tâches applicatives activées, ainsi que du score de qualité correspondant. Pour notre cas d'étude, la liste des paramètres et de leurs valeurs possibles est introduite Table 1. Le nombre maximum théorique de modes d'exécutions est au égal au produit des nombres de valeurs pour chaque paramètre. Ceci donne un total de 24 arrangements de paramètres pour lesquels on calcule au moins un ordonnancement de tâches sur les RRs.

Paramètres	Positions	Valeurs
R - Résolution d'image	{360p ; 480p}	{0,5 ; 1}
I - Images par seconde	{30ips ; 60ips}	{30 ; 60}
F - Filtrage	{Non ; Sobel ; Médian}	{0 ; 0,75 ; 1}
C - Chiffrement	{Non ; AES}	{0 ; 1}

TABLE 1 – Liste des paramètres influant sur la qualité, de leurs positions fonctionnelles et valeurs numériques possibles

Une fois les modes d'exécutions définis, une fonction objectif doit être introduite pour quantifier le score de qualité de chaque mode. Cette fonction, relative aux préférences de l'utilisateur, est une entrée de notre modèle de qualité. Elle peut être définie par une formule mathématique (eg : rapport signal à bruit) ou obtenue empiriquement (*quality assessment*). La fonction objectif Q de notre encodeur a été extrapolée et définie empiriquement à titre d'exemple à partir d'une étude de préférences d'utilisateurs sur les paramètres vidéos [7] à laquelle nous ajoutons un bonus pour le filtrage et le chiffrement. Avec les valeurs de R, I, F et C de la Table 1 :

$$Q(R,I,F,C) = \frac{0,75R + 2,5.I.10^{-2} + F + C}{4,25} \quad (1)$$

4.2 Sensibilité aux paramètres

Les paramètres définissant les modes d'exécution impactent le graphe de tâche de l'application visée de différentes manières. Certains paramètres peuvent impacter fortement cer-

taines tâches et non d'autres. Par exemple, modifier le paramètre Filtrage introduit un changement fonctionnel complet de la tâche T0, mais n'impacte pas les tâches du noyau de l'encodeur H264. Les tâches du noyau sont donc fonctionnellement insensibles au paramètre Filtrage.

Certains paramètres peuvent introduire des changements non pas au niveau des tâches, mais au niveau de l'application en elle-même. C'est notamment le cas du paramètre Images par seconde dans notre cas d'usage. Ce dernier n'influe pas fonctionnellement et individuellement les tâches, mais réduit le temps alloué à l'application pour encoder une image. L'ensemble des tâches sont alors fonctionnellement insensibles à ce paramètre, mais un changement sur ce dernier peut invalider un partitionnement et ordonnancement déjà implémenté sur la plate-forme.

Les sensibilités aux paramètres de qualité peuvent être observées à partir des modes d'exécution du graphe de tâche. En effectuant la différence de graphe de tâches de deux modes d'exécutions différents d'un seul paramètre, on obtient les tâches sensibles à ce paramètre ne pouvant bénéficier de réutilisation.

5 Méthode

5.1 Gestionnaire hybride orienté qualité

En partitionnant une application sur les RRs à l'aide d'un algorithme d'ordonnancement tel que [8], on obtient la liste des tâches à exécuter par RR ainsi que leur ordre d'exécution. Associé au score de qualité du mode d'exécution partitionné, ce résultat est appelé *solution* dans la suite de cet article.

Dans nos travaux, nous avons développé un gestionnaire hybride [9] permettant l'obtention de solutions au temps de la compilation, afin de les stocker dans la mémoire RAM en vue d'une utilisation au temps de l'exécution. En ligne, notre heuristique Q-Greedy cherche une solution dans cette base de donnée en maximisant le score de qualité. L'intérêt de cette méthodologie est de disposer d'un grand nombre de solutions permettant d'en changer lorsque des RRs sont contraintes par d'autres tâches, ou en cas de panne [10]. Cette méthodologie évite le calcul d'un ordonnancement en ligne, ce problème étant NP-difficile. Une vue d'ensemble de cette méthodologie est présentée Figure 3.

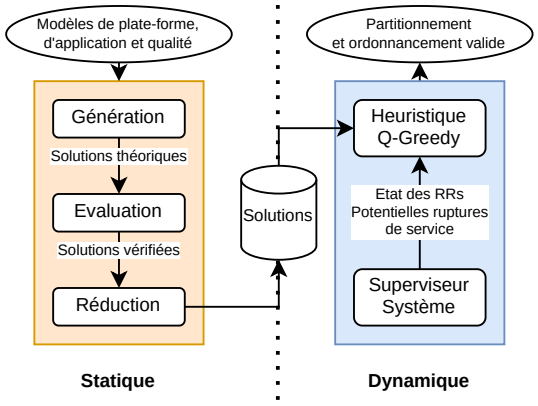

FIGURE 3 – Vue d'ensemble du gestionnaire hybride. La partie statique s'exécute hors-ligne, et la partie dynamique en ligne.

La génération de solutions consiste à obtenir un grand nombre de partitionnements par combinaison linéaire. Les ordonnancement des solutions générés sont évalués pour s'assurer qu'ils ne comprennent pas de blocages, et obtenir le temps d'exécution de ce partitionnement. On calcule ensuite le score qualité de la solution en fonction des paramètres de cette solution. Enfin, nous réduisons le nombre de solutions obtenues en supprimant celles ne respectant pas des contraintes relatives à l'application ciblée. Par exemple, celles dont le temps d'exécution dépasse la deadline temporelle (33ms pour 30ips) pour notre encodeur.

5.2 Recherche de solutions avec réutilisation

Lors de l'exécution, l'heuristique Q-Greedy cherche dans la base de donnée de solution. Cette dernière est un ensemble de solution ψ regroupées par modes d'exécution $\mathcal{M}$ dans un sous-ensemble $\psi(\mathcal{M}_\rangle$. Ces modes sont ordonnés par ordre croissant de score qualité. La recherche est déclenchée à chaque fin d'exécution d'une itération de l'application, soit une image dans le cas de notre encodeur, par le superviseur système. Le superviseur peut déclencher l'heuristique en mode *Dégradé* lorsqu'il constate que le système est trop contraint, et ne considère alors que les modes d'exécution de qualité au plus égales à celle du mode courant $\mathcal{M}^c$. Les contraintes γ_i sont des temps pendant lesquels une RR i est occupée et les tâches de l'application ne peuvent s'exécuter. L'heuristique est présenté ci-dessous :

Algorithme 1 : Heuristique de recherche Q-Greedy

Entrées : Set de solutions ψ, de modes d'exécution $\mathcal{M}$, solution courante S^c, mode d'exécution courant $\mathcal{M}^c$, contraintes sur les RRs γ
Sorties : Nouvelle solution

1 **si** *Dégradé* **alors** $\mathcal{M}_i \leftarrow \mathcal{M}^c$; $Q(S^c) = 0$
2 **sinon** $\mathcal{M}_i \leftarrow$ *mode d'exécution de plus haut score* Q
3 **tant que** $\mathcal{M}_i$ *existe* **faire**
4 **si** $\mathcal{M}_i \neq \mathcal{M}^c$ **alors** *chercher les tâches réutilisables*
5 *Trier* $\psi(\mathcal{M}_i)$ *par tâches réutilisables en premier*
6 **pour** $S_i \in \psi(\mathcal{M}_i)$ **faire**
7 *vérifier la faisabilité de* S_i *compte tenu de* γ
8 **si** $Q(S) > Q(S^c)$ **alors retourner** S_i
9 $i \leftarrow i - 1$ // Passer au mode suivant

La recherche de solution est biaisée vers les solutions réutilisant des tâches déjà implémentées. La réutilisation peut s'opérer si une tâche se retrouve instanciée seule sur une RR. Auquel cas, une telle tâche étant insensible au changement de mode est réutilisable d'un partitionnement à l'autre. Ce biais est introduit dans l'heuristique en triant le sous-ensemble étudié pour présenter en premier les solutions avec des tâches réutilisables.

Les solutions sont étudiées en vérifiant si les partitionnements restent viable compte tenu des contraintes γ sur la disponibilité des RRs. Si l'heuristique ne trouve pas de solution

viable, il dégrade le mode d'exécution en passant au mode non étudié avec le plus haut score Q. Enfin, si la solution est faisable et que sa qualité est supérieure à celle de la solution courante, elle est choisie comme nouveau partitionnement.

6 Expérimentations

La méthodologie a été validée sur une cible comportant 4 RRs. Seul le superviseur système de la Figure 3 a été simulé par un générateur de contraintes, simulant des tâches fictives imposant des restrictions totales ou partielles sur les RRs.

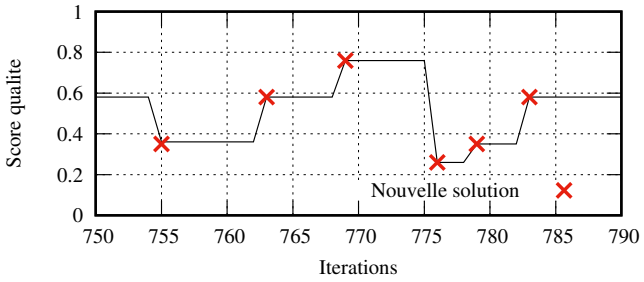

FIGURE 4 – Évolution du score qualité au fil des itérations de l'application. Les croix indiquent les itérations où l'algorithme applique un nouveau partitionnement des tâches sur les RRs.

#	Mode d'exécution	RR0	RR1	RR2	RR3
755	480p, 30ips	T4	T2	T3	T1, T5
763	480p, 30ips et AES	**T4**	T2, T6	**T3**	T1, T5
769	480p, 30ips, Sobel et AES	**T4**	T2, T6	T0, T3	T1, T5
776	360p, 30ips	T1	T3, T5	T4	T2

TABLE 2 – Nouvelles implémentations particulières où la réutilisation a lieu, par numéro d'itérations. Les tâches T3 et T4 en gras ont été réutilisées entre lors des changements de modes.

La Figure 4 présente l'évolution du score qualité au cours de l'expérience. Ce score évolue selon les implémentations choisies dans la Table 2. À l'itération 763, l'application s'enrichit du chiffrement avec T6. Le score qualité augmente, et les tâches T4 et T3 peuvent être réutilisés. La même situation arrive à l'itération 769 en s'enrichissant de la tâche de filtrage Sobel. Le système étant un peu plus contraint par cette nouvelle tâche, il ne peut réutiliser T3, mais garde T4. Finalement, le système doit dégrader l'application à l'itération 776 pour garantir son exécution, ne pas pas réutiliser de tâches.

L'expérience a été lancée sur 100k itérations d'activations du superviseur système simulé. La réutilisation est effective dans 12% des cas pour une perte en score qualité de 4% en moyenne par rapport à l'optimal, pour un temps de décision de Q-Greedy moyen de 3ms. La réutilisation permet cependant d'économiser les accès à la mémoire et de réduire la latence lors de changement de modes d'exécutions. Cette réduction peut monter jusqu'à 6ms par RM réutilisé, soit le temps de reconfiguration maximal d'une RR pour l'architecture considérée.

7 Conclusion

Nous avons présenté une amélioration de notre méthodologie d'optimisation de qualité d'expérience pour application sur plate-forme à base de FPGA dynamiquement reconfigurable [9]. Cette amélioration consiste en un biais de l'heuristique de recherche de solution au temps de l'exécution envers une réutilisation de modules reconfigurables déjà en place sur FPGA. Les expérimentations sur la plate-forme simulée montrent que l'heuristique est capable de conserver les implémentations de tâches sur RRs entre changements de partitionnement.

Cette amélioration est particulièrement adaptée pour les applications type flot de données à gros grains implémentées sur ce type d'architecture [11]. L'approche peut être généralisée à tout type d'applications sujette à des paramètres qualité impactant seulement une partie des tâches qui la compose.

Références

[1] Nurvitadhi, E. et al., "Can FPGAs beat GPUs in accelerating next-generation deep neural networks ?", FPGA, 2017.

[2] Dorflinger, A. et al., "Hardware and software task scheduling for ARM-FPGA platforms", AHS, 2018.

[3] Vipin, K. & Fahmy, S., "FPGA dynamic and partial reconfiguration : a survey of architectures, methods, and applications", ACM Computing Surveys, 2018.

[4] Vaishnav, A., Pham, K., Powell, J. & Koch, D., "FOS : a modular FPGA operating system for dynamic workloads", ACM TRETS, 2020.

[5] Sultana, B. et al., "VR-ZYCAP : a versatile resource-level ICAP controller for ZYNQ SOC", Electronics, 2021.

[6] Theelen, B. et al., "A scenario-aware data flow model for combined long-run average and worst-case performance analysis." MEMOCODE, 2006.

[7] Debattista, K. et al., "Framerate vs resolution : a subjective evaluation of perceived quality under varying computational budgets." Computer Graphics Forum, 2017.

[8] Roy, S., Devaraj, R., Sarkar, A. & Senapati, D., "SLAQA : quality-level aware scheduling of task graphs on heterogeneous distributed systems", ACM Transactions on Embedded Computing Systems, 2021.

[9] Duhamel, A. & Pillement, S., "QoS aware design-time/run-time manager for FPGA-based embedded systems" DASIP, 2022.

[10] Sahoo, S., Nguyen, T., Veeravalli, B. & Kumar, A., "Multi-objective design space exploration for system partitioning of FPGA-based dynamic partially reconfigurable systems", Integration, 2019.

[11] Jain, A., Maskell, D. & Fahmy, S., "Are coarse-grained overlays ready for general purpose application acceleration on FPGAs ?", DASC, 2016.