Exploration architecturale d'accélérateur pour des architectures multi-cœurs hétérogènes

Baptiste ROUX[1], Matthieu GAUTIER[1], Olivier SENTIEYS[1],

[1]Univ Rennes, CNRS, IRISA, 6 rue de Kerampont, 22 300, Lannion, France

baptiste.roux@inria.fr, matthieu.gautier@irisa.fr,olivier.sentieys@inria.fr

Résumé – La programmation d'architectures multiprocesseurs hétérogènes combinant plusieurs cœurs de processeur et accélérateurs matériels est un véritable défi. En particulier, la plupart des méthodologies d'exploration de l'espace de conception ne prend pas en compte la consommation d'énergie en raison de la difficulté d'obtenir rapidement et précisément une estimation. Dans ce but, cet article propose une méthode d'exploration reposant sur une formulation hybride de l'énergie (analytique et expérimentale) résolue à l'aide de la programmation linéaire à nombres entiers mixtes. L'approche est validée sur deux noyaux d'applications utilisant une architecture Zynq montrant la précision de la formulation.

Abstract – Programming heterogeneous multiprocessor architectures combining multiple processor cores and hardware accelerators is a real challenge. Moreover, energy consumption is not well supported in most of design space exploration methodologies due to the difficulty to estimate energy consumption fast and accurately. To this aim, this paper proposes and validates an exploration method that relies on a hybrid energy formulation (both analytical and experimental) solved using Mixed Integer Linear Programming. The approach is validated on two application kernels using Zynq-based architecture showing the accuracy of the formulation.

1 Introduction

Cette dernière décennie a vu l'avènement des nouvelles architectures multi-coeurs hétérogènes (HMpSoC - Heterogeneous Multiprocessor System-on-Chip) combinant des processeurs généralistes (SW) et des accélérateurs matériels (HW) [1]. Outre le gain en performance attendu, ces architectures ont induit de nouvelles problématiques en termes de flot de conception. En effet, avec la multiplication du nombre et du type d'organes de calcul, l'espace d'exploration est devenu immense. Ainsi, lorsque la consommation devient un facteur clé, il est essentiel de pouvoir explorer rapidement cet espace pour obtenir des solutions efficaces en énergie [2].

Cet article présente une méthode d'exploration de l'espace de conception d'accélérateurs matériels implémentés sur une architecture HMpSoC. Cette méthode s'appuie sur un modèle de consommation hybride associant expressions analytiques et mesures expérimentales [3] et sur une formulation de programmation linéaire en nombre entier mixte (MILP - Mixed Integer Linear Programming). Ce problème d'optimisation permet de sélectionner efficacement les types d'accélérateurs HW et le partitionnement HW/SW et ainsi d'obtenir une implémentation efficace en énergie pour une application tuilée [4].

Le flot d'exploration est présenté dans la Section 2 ainsi que la modélisation de l'énergie. La Section 3 introduit la méthode d'exploration de l'espace de conception. Enfin, la Section 4 donne les performances de la méthode pour deux applications.

2 Flot d'exploration orienté énergie

2.1 Modèles d'architectures hétérogènes

Nous considérons des architectures hétérogènes composées de N_{SW} cœurs de calcul SW et de N_{HW} cœurs HW. Les cœurs SW peuvent s'appuyer sur la même architecture de processeur ou sur un ensemble de cœurs différents. Les cœurs HW sont implémentés dans la partie PL (Programmable Logic) de l'architecture. La PL peut être partitionnée en un maximum de N_{HW} cœurs HW indépendants. Cette restriction, basée sur le nombre maximum de transferts de données simultanés de la PL vers/depuis la mémoire principale, empêche l'apparition d'une congestion mémoire entrainant un temps d'accès mémoire et un coût énergétique imprévisibles. De plus, toutes les ressources utilisées par les cœurs doivent être inférieures aux ressources globales disponibles dans l'architecture. Utilisée dans nos expérimentations, l'architecture Zynq de Xilinx est composée de deux cœurs SW identiques et d'une partie PL FPGA. Les cœurs HW communiquent avec les cœurs SW via quatre ports dédiés HP (High Performance) de la mémoire DDR (Double-Data-Rate).

2.2 Objectifs du DSE

La méthode d'exploration de l'espace de conception pour les architectures HMpSoC est présentée dans la Figure 1. Elle permet d'optimiser la partitionnement HW/SW pour différentes contraintes utilisateurs, en particulier la contrainte énergétique. Le flot cible les applications parallèles tuilées, afin de s'abstraire des étapes d'extraction du parallélisme, et repose sur un modèle énergétique analytique, dont les paramètres énergétiques sont extraits par des micro-benchmarks [3] .

La transformation en tuile [5] est une technique de parallélisation qui offre des avantages intéressants pour la conception d'accélérateurs : l'augmentation de la réutilisation des données dans les mémoires tampon ; l'exposition de parallélismes gros grain exploités dans le partitionnement HW/SW ; l'exposition d'un parallélisme grain fin dans chaque tuile pouvant être utilisé pour concevoir efficacement l'architecture HW.

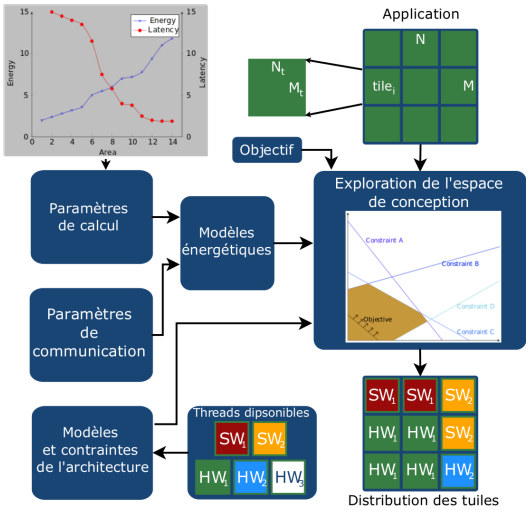

FIGURE 1 – Vue globale de l'exploration proposée.

Dans cette étude, nous considérons des applications 2D avec un ensemble de N_{tiles} tuiles (tiles en anglais) indépendantes réparties parmi les cœurs d'exécution de l'architecture cible. L'application tuilée est représentée en entrée du flot de conception de la Figure 1 alors que l'allocation des tuiles aux ressources HW et SW est la sortie du flot.

Le but de la méthode d'exploration proposée est de trouver la meilleure configuration qui minimise l'objectif de l'utilisateur. Une configuration est un vecteur $\vec{C}$ composé de la distribution des tuiles entre les cœurs SW et HW et du type d'implémentation utilisé pour les cœurs HW. Un exemple d'une telle distribution est représenté sur la Figure 1 en sortie du DSE. Le nombre de tuiles allouées au cœur SW i et au cœur HW j sont notés respectivement $tiles^i_{SW}$ et $tiles^j_{HW}$. Les cœurs HW peuvent utiliser un bloc de calcul parmi un ensemble d'implémentations disponibles qui ont des performances différentes en surface, en énergie et en latence (comme illustrées dans la Figure 1 en haut à gauche où chaque point correspond à une implémentation). Dans notre étude, l'exploration matérielle a été effectuée manuellement à l'aide des outils de synthèse de haut niveau. Le bloc de calcul est synthétisé sous différentes contraintes de latence pour obtenir différentes implémentations variant de 1 à N_{impl}, avec $impl_j$ représentant l'implémentation utilisée dans le cœur HW j. Une configuration $\vec{C}$ est donc définie par :

$$\vec{C} = [tiles^1_{SW}; \ldots; tiles^{N_{SW}}_{SW}; (tiles^1_{HW}, impl_1); \ldots; (tiles^{N_{HW}}_{HW}, impl_{N_{HW}})]. \quad (1)$$

L'objectif du DSE est donc de trouver la meilleure configuration $\vec{C}_{opt}$ qui minimise le coût (latence, énergie) tel que :

$$\vec{C}_{opt} = \min_{\forall \vec{C}} Cost(\vec{C}), \quad (2)$$

avec $Cost(\bullet)$ la fonction de coût définie par des modèles analytiques introduits dans la partie suivante.

2.3 Modèles énergétiques et du temps d'exécution

La consommation d'énergie d'une application exécutée sur une architecture HMpSoC dépend de trois sources principales : l'énergie statique dissipée pendant le temps d'exécution, la consommation d'énergie dynamique utilisée pour les calculs et l'énergie utilisée pour les communications entre les cœurs de calcul. Les applications sont composées de N_{tiles} tuiles indépendantes dont l'énergie et le temps nécessaires à son calcul ne dépendent que du cœur d'exécution ciblé. La quantité de données requises pour le calcul des tuiles est supposée connue au moment de l'exploration. Les cœurs HW sont gérés avec des appels de bas niveau et des interruptions qui sont encapsulées dans un thread SW. La répartition des threads entre les cœurs de calcul est effectuée séquentiellement et commence par les cœurs HW afin de minimiser le nombre de changements de contexte.

Temps de calcul Pour une configuration $\vec{C}$, le temps total de calcul $T_t(\vec{C})$ est le temps maximal entre T_{SW} et T_{HW}, correspondant respectivement aux temps de calcul sur les cœurs SW et HW, déterminés par :

$$T_{HW}(\vec{C}) = \max_{j \in (1 \ldots N_{HW})} \left[T^{impl_j}_{comp} \times tiles^j_{HW} + j \times T_{spawn} \right] \quad (3)$$

$$T_{SW}(\vec{C}) = \max_{i \in (1 \ldots N_{SW})} \left[T^{SW_i}_{comp} \times tiles^i_{SW} + (N^{used}_{HW}(\vec{C}) + i) \times T_{spawn} \right] \quad (4)$$

avec $T^{impl_j}_{comp}$ and $T^{SW_i}_{comp}$ le temps de calcul d'une tuile exécutée respectivement sur l'implémentation HW j et sur le cœur SW i. T_{spawn} représente le temps nécessaire à configurer le thread de calcul. $N^{used}_{HW}(\vec{C})$ est le nombre de cœurs HW utilisés pour la configuration $\vec{C}$.

Consommation d'énergie L'énergie totale $E_t(\vec{C})$ consommée par l'exécution d'une application tuilée avec la configuration $\vec{C}$ est composée de l'énergie statique et de l'énergie dynamique due aux calculs et aux communications :

$$E_t(\vec{C}) = E_{stat} + \sum_{i=1}^{N_{SW}} (E^{SW_i}_{comp} + E^{SW}_{com}) \times tiles^i_{SW} + \sum_{j=1}^{N_{HW}} (E^{impl_j}_{comp} + E^{HW}_{com}) \times tiles^j_{HW}, \quad (5)$$

avec E_{stat} l'énergie statique consommée, E^{SW}_{com} et E^{HW}_{com} les énergies des communications requises pour l'exécution d'une tuile sur un cœur SW et HW respectivement. $E^{SW_i}_{comp}$ et $E^{impl_j}_{comp}$ sont les énergies nécessaires au calcul d'une tuile sur le cœur SW i et sur l'implémentation HW $impl_j$.

Extraction des paramètres de communications et de calculs La méthode décrite dans [3] est utilisée pour estimer l'énergie des communications entre les cœurs E^{SW}_{com} et E^{HW}_{com} et l'énergie de calcul d'une tuile ($E^{SW_i}_{comp}$ ou $E^{impl_j}_{comp}$).

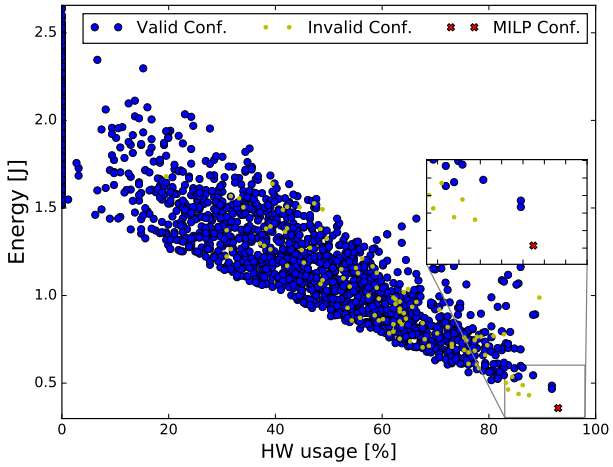

FIGURE 2 – Exploration de l'espace de conception pour l'application Matmult.

3 Exploration de l'espace de conception

3.1 Espace de conception

En considérant l'architecture cible similaire à l'architecture Zynq avec deux cœurs SW du même type et quatre cœurs HW, l'espace de conception est décrit comme un polyèdre et sa taille peut être calculée à l'aide de la bibliothèque `isl` [6] en fonction de N_{impl} et N_{tiles}. Par exemple, avec $N_{impl} = 3$ et $N_{tiles} = 256$, l'espace de conception est composé de $7\,866 \times 10^{11}$ points. En supposant que la consommation d'énergie peut être obtenue en environ 1 ms, une recherche exhaustive nécessiterait plus de 9 jours et n'est donc pas une approche valide.

La Figure 2 montre un sous-ensemble choisi au hasard des configurations de l'espace de conception pour l'application multiplication de matrice (Matmult) étudiée par la suite. La figure représente la consommation d'énergie en fonction du pourcentage d'utilisation HW, correspondant à la proportion des tuiles calculée sur la partie HW. Cette figure montre que l'espace de conception est très grand et que l'exploration ne garantit pas la validité d'implémentation des configurations obtenues. Une configuration non valide est une configuration qui nécessite davantage de ressources que celles disponibles sur le circuit.

Pour résoudre ce problème, la section suivante présente une méthode d'optimisation basée sur la formulation MILP, qui résout le problème défini par (2) et trouve la meilleure solution dans l'espace de conception valide. La Figure 2 donne également la solution obtenue grâce à l'optimisation MILP.

3.2 Formulation MILP

La programmation MILP est une solution connue pour la résolution des problèmes de partitionnement. Avec cette approche, les contraintes sont définies comme un ensemble d'inégalités avec des variables booléennes, entières (discrètes) et non-entières (continues). Les solutions peuvent être ensuite déterminées en utilisant des solveurs commerciaux ou open source.

Cette section présente les inégalités décrivant les contraintes de notre problème. Ensuite, deux fonctions de coût sont définies.

Contraintes du modèle Trois contraintes sont définies afin d'obtenir une configuration valide :

- Contrainte de couverture : elle permet de garantir que chaque tuile ne sera exécutée qu'une seule fois :

$$N_{tiles} = \sum_{i=1}^{N_{SW}} tiles_{SW}^{i} + \sum_{j=1}^{N_{HW}} tiles_{HW}^{j}. \quad (6)$$

- Contrainte d'unicité : pour s'assurer qu'une seule implémentation matérielle est utilisée pour chaque thread HW, la variable $usedImpl_{type}^{j}$ est définie par :

$$usedImpl_{type}^{j} = \begin{cases} 1, & \text{si } impl_j = type \\ 0, & \text{sinon} \end{cases} \quad (7)$$

La contrainte d'unicité garantit qu'une seule implémentation est utilisée dans chaque cœur HW :

$$\forall j \in (1 \dots N_{HW}) : \sum_{type=1}^{N_{impl}} usedImpl_{type}^{j} \leq 1. \quad (8)$$

- Contrainte de ressource : les ressources disponibles de la partie HW (PL) sont séparées en quatre catégories : les blocs RAM (BRAM), les blocs DSP (DSP), les Flip-Flops (FF) et les Look Up Tables (LUT). La contrainte de ressource garantit la disponibilité des ressources :

$$\forall r \in \{\text{BRAM, DSP, FF, LUT}\} :$$
$$\sum_{j=1}^{N_{HW}} \sum_{type=1}^{N_{impl}} usedImpl_{type}^{j} \times RscCost_{r}^{type} \leq avlbRsc_r, \quad (9)$$

avec $RscCost_{r}^{type}$ le coût d'une ressource r pour l'implémentation $type$ et $avlbRsc_r$ les ressources disponibles de type r de l'architecture.

Fonctions de coûts Le solveur construit d'abord le vecteur de configuration $\vec{C}$ en utilisant les contraintes, puis sélectionne la meilleure solution en utilisant une fonction de coût dépendant de l'objectif d'optimisation. Deux objectifs sont définis : minimiser l'énergie d'exécution globale ou minimiser le temps d'exécution global. Ces fonctions objectif sont définies par :

$$\vec{C}_{energy} = \min_{\forall \vec{C}} \left[E_t(\vec{C}) \right] \quad (10)$$

$$\vec{C}_{time} = \min_{\forall \vec{C}} \left[\max \left[T_{SW}(\vec{C}), T_{HW}(\vec{C}) \right] \right] \quad (11)$$

L'optimisation MILP donne le meilleur vecteur de configuration en fonction de l'objectif et une estimation de ses caractéristiques (consommation d'énergie et temps d'exécution). La Figure 2 donne la solution obtenue avec la méthode MILP pour la multiplication de matrice. Cette solution est toujours optimale en termes de coûts et est calculée en un temps limité.

Conf.		Thread ID					
		SW0	SW1	HW0	HW1	HW2	HW3
Mat.	Impl.	SW	SW	LnP 248	LnP 248	LnP 148	none
	N_{tiles}	9	9	87	85	66	0
Stenc.	Impl.	SW	SW	LnP 114	LnP 114	none	none
	N_{tiles}	25	24	104	103	0	0

TABLE 1 – Configurations obtenues par l'optimisation MILP.

Application		Time [s]	err. [%]	Energy [J]	err. [%]
Mat.	Est.	2.29×10^{-1}	3.35%	3.558×10^{-1}	5.03%
	Meas.	2.369×10^{-1}		3.746×10^{-1}	
Sten.	Est.	1.121×10^{-1}	9.58%	1.566×10^{-1}	10.25%
	Meas.	1.24×10^{-1}		1.745×10^{-1}	

TABLE 2 – Précision de l'exploration MILP.

4 Performances de l'exploration MILP

4.1 Applications ciblées

Le premier noyau est une application de multiplication matricielle (Matmult). Les matrices d'entrée et de sortie sont de taille 512×512. Le calcul matriciel est divisé en 256 tuiles de taille 32×32. Chaque tuile lit des lignes ou des colonnes entières des matrices d'entrée et écrit les résultats dans la tuile de sortie. Le deuxième noyau est un calcul de Stencil basé sur un filtre de taille 4×4, qui est successivement appliqué 10 fois. Les matrices d'entrée et de sortie sont de taille 542×542 et 512×512, respectivement. L'application est tuilée avec un chevauchement afin d'empêcher la dépendance entre les tuiles, ce qui entraîne le calcul de 256 tâches indépendantes.

4.2 Optimisation MILP

Dans un premier temps, les paramètres de coûts énergétiques des communications entre les cœurs et des différentes implémentations SW et HW sont extraits pour l'architecture Zynq. Ces modèles sont ensuite intégrés dans la formulation MILP pour les 2 applications et le problème d'optimisation est résolu pour la fonction de coût énergétique (10) à l'aide du solveur MILP *Gurobi* [7] sur un processeur *Intel i7 Haswell-ult* fonctionnant à 2.10GHz. La solution MILP est obtenue après 18980 itérations du simplexe en $\simeq 0.73$ s pour l'application Matmult et après 348 itérations en $\simeq 0.03$ s pour l'application Stencil. Cette différence est principalement due au plus grand nombre d'implémentations HW disponibles pour l'application Matmult (6 pour Matmult contre 3 pour Stencil).

La Table 1 donnent les configurations MILP obtenues avec la répartition des tuiles sur les cœurs et les implémentations utilisées. Pour l'application Matmult, 18 tuiles sont réparties équitablement sur les cœurs SW, les tuiles restantes sont réparties sur trois noyaux HW. L'implémentation *LnP 248* est utilisée sur deux cœurs HW, le troisième utilise une implémentation plus petite pour occuper l'espace restant dans la partie PL du FPGA. Pour l'application Stencil, seuls deux cœurs HW sont utilisés, les implémentations HW du Stencil utilisant plus de ressources que celles de Matmult.

4.3 Précision de l'estimation

Afin de valider l'approche proposée, les estimations du temps d'exécution et de l'énergie consommée de la configuration MILP sont comparées aux mesures réelles de la même configuration sur la carte Zynq. Les valeurs mesurées sont des moyennes sur 40 exécutions indépendantes de la configuration MILP. La Table 2 donne les valeurs estimées et mesurées pour les deux applications. L'erreur moyenne pour l'application Matmult est d'environ 5 %, ce qui est relativement faible pour des mesures de puissance sur une architecture réelle. Pour l'application Stencil, l'erreur moyenne est plus élevée. Son temps d'exécution étant plus court que celui de la multiplication de matrice, l'application Stencil est plus sensible à l'imprécision de la mesure de puissance.

5 Conclusion

A partir d'un modèle de consommation calculable rapidement des architectures multi-cœurs hétérogènes, nous avons proposé dans cet article une formalisation MILP, permettant de trouver la configuration optimale en terme d'énergie pour le placement d'applications tuilées sur ce type d'architecture. Pour deux applications, nous avons montré la rapidité de calcul de la méthode ainsi que sa précision.

Références

[1] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark Silicon and the End of Multicore Scaling," in *International Symposium on Computer Architecture (ISCA)*, June 2011, pp. 365–376.

[2] R. B. Atitallah, E. Senn, D. Chillet, M. Lanoe, and D. Blouin, "An Efficient Framework for Power-Aware Design of Heterogeneous MPSoC," *IEEE Transactions on Industrial Informatics*, Feb. 2013.

[3] B. Roux, M. Gautier, O. Sentieys, and S. Derrien, "Communication-based power modelling for heterogeneous multiprocessor architectures," in *IEEE 10th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (mcSoC)*, Sept 2016, pp. 209–216.

[4] B. Roux, M. Gautier, O. Sentieys, and J.-P. Delahaye, "Poster : Fast and Energy-Driven Design Space Exploration for Heterogeneous Architectures," in *IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2017.

[5] M. E. Wolf and M. S. Lam, "A Loop Transformation Theory and an Algorithm to Maximize Parallelism," *IEEE Transactions on Parallel and Distributed Systems*, vol. 2, no. 4, pp. 452–471, October 1991.

[6] S. Verdoolaege, *isl : An Integer Set Library for the Polyhedral Model*. Springer Berlin Heidelberg, 2010.

[7] Gurobi Optimization, 2016. Available : `http://www.gurobi.com`