SCAPE : Regroupement d'Acteurs flux de Données en Fonction de l'Architecture pour une Complexité d'Ordonnancement Contrôlée [1]

Ophélie RENAUD Karol DESNOS Jean-François NEZAN
Univ Rennes, INSA Rennes, CNRS, IETR – UMR 6164, F-35000 Rennes

Résumé – Les modèles flux de données sont des paradigmes de programmation efficaces pour exposer le parallélisme d'une application. Traditionnellement, les méthodes de placement et d'ordonnancement pour les modèles flux de données reposent sur des transformations complexes de graphes pour exposer tout le parallélisme potentiel d'une application, ce qui peut donner lieu à des graphes complexes pour des applications hautement parallèles. La complexité des algorithmes de placement et d'ordonnancement sur ces graphes complexes est prohibitive, alors que le parallélisme exposé dépasse souvent le parallélisme potentiel de l'architecture cible. Nous proposons dans cet article la méthode automatique SCAPE pour contrôler la complexité de la transformation du graphe de pré-ordonnancement en utilisant des informations provenant des modèles d'architecture et d'application. En diminuant la complexité de ce graphe, les algorithmes de placement et d'ordonnancement sont accélérés au détriment potentiel de l'ordonnancement produit. La méthode SCAPE permet de contrôler la taille du graphe de pré-ordonnancement pour trouver le compromis entre rapidité et qualité du placement-ordonnancement. La méthode SCAPE accélère les algorithmes de placement et de l'ordonnancement de l'état de l'art d'un facteur pouvant aller jusqu'à 2 ordres de grandeur.

Abstract – Dataflow models are efficient programming paradigms for expressing the parallelism of an application. Traditionally, mapping and scheduling methods for dataflow models rely on complex graph's transformations to explicit their parallelism which can result in a complex graph for embarrassingly parallel applications. For such applications, state-of-the-art mapping and scheduling techniques are prohibitively complex, while the exposed parallelism often exceeds the parallel processing capabilities of the target architecture. We propose SCAPE, an automated method to control the complexity of the pre-scheduling graph transformation by using information from the architecture and application models. By decreasing the complexity of the graph, the mapping scheduling task is accelerated at the potential expense of the produced schedule. Our method offers a limited and controlled decrease of the schedule quality while enabling mapping and scheduling execution time between 1 and 2 orders of magnitude faster than state-of-the-art techniques.

1 Introduction

La technologie de traitement des signaux numériques est apparue dans les années 1960 et s'est développée rapidement, devenant plus complexe au fil des ans, en particulier avec l'arrivée des applications d'apprentissage automatique il y a une dizaine d'années. Pour répondre aux besoins toujours croissants de puissance de calcul et de rapidité d'exécution de ces applications, les développeurs ont d'abord cherché à augmenter la fréquence des éléments de calcul (PE) individuels, puis se sont tournés vers les systèmes embarqués hétérogènes à plusieurs cœurs.

L'exploitation optimisée du parallélisme maximal de ces architectures multicœurs est un défi de taille. Le développement d'un code parallèle est fastidieux et n'est pas adapté à la gestion des mises à jour matérielles et logicielles pendant la phase d'exploitation du projet. Des outils tels que Simulink [5] et Xilinx AI Xilinx AI Engine Technology [1] sont alors étudiés pour automatiser le déploiement rapide de nouveaux algorithmes sur des systèmes informatiques. Les deux outils utilisent une approche de flux de données qui implique la représentation des algorithmes sous forme de graphes. Dans ces graphes, les nœuds, appelés acteurs, symbolisent les calculs effectués, tandis que les arcs dirigés, appelés mémoires tampon en file *premier entré, premier sorti* (FIFO), représentent les échanges de données entre les nœuds.

La génération automatique de code parallèle à partir de ces modèles flux de données nécessite la résolution de plusieurs problèmes NP-Complets, en particulier pour l'allocation des ressources matérielles. Les calculs sont distribués sur les PE de l'architecture cible et lisent et écrivent, pendant l'exécution d'une application, sur des mémoires tampons FIFO assignés à une gamme d'adresses mémoire. Les choix d'allocation des ressources peuvent être faits au moment de la compilation ou au moment de l'exécution. L'allocation au moment de l'exécution entraîne une surcharge de travail sur les PE se répercutant négativement sur les performances de l'application. Pour ces raisons, ce document étudie les méthodes qui allouent les ressources au moment de la compilation, pendant la synthèse du logiciel. Le processus de synthèse logicielle est responsable de la traduction d'un modèle flux de données en un prototype exécutable.

Les méthodes classiques d'allocation des ressources comportent deux phases : le *placement* consiste à répartir des acteurs sur les PE. L'*ordonnancement* consiste à ordonner l'exécution des acteurs sur les PE. Le temps nécessaire au processus de placement et d'ordonnancement croît de manière exponentielle avec le nombre de PE, le nombre de nœuds et d'arêtes du graphe flux de données [6].

Ce document présente la méthode mise à l'échelle de groupements d'acteurs sur le nombre d'élément de traitement (SCAPE). Il s'agit d'une méthode de groupement basée sur la hiérarchie qui transforme une application afin d'adapter son

[1] Ce travail a été soutenu par DARK-ERA (ANR-20-CE46-0001-01).

degré de parallélisme aux capacités de calcul parallèle de l'architecture ciblée. La méthode offre autant de configurations de regroupement qu'il y a de niveaux hiérarchiques dans le graphe d'entrée flux de données synchrone (SDF), ce qui permet à l'utilisateur de choisir la granularité requise pour réduire le temps de synthèse du logiciel.

La section 2 présente les modèles flux de données, la méthode standard de placement et d'ordonnancement et l'état de l'art des heuristiques de regroupement d'acteur. La section 3 décrit la méthode proposée et l'ossature du code qui en résulte. La section 4 présente les résultats expérimentaux sur plusieurs configuration de groupements montrant le compromis entre le temps d'exploration de l'espace de conception et le temps de latence de l'ordonnancement produit. Enfin la section 5 conclut ce document.

2 Contexte et travaux connexes

2.1 Modèles flux de données synchrones

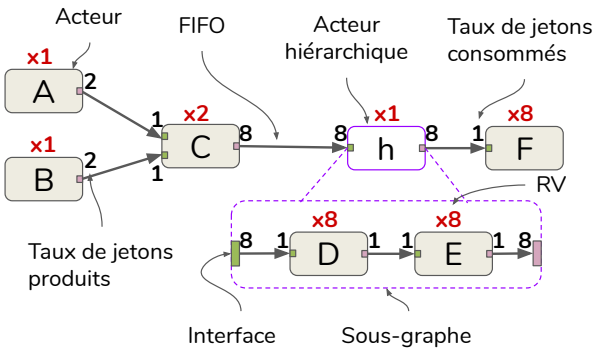

FIGURE 1 : Sémantique du modèle PiSDF

Le modèle flux de données le plus étudié est le SDF [7] dans lequel les nombres entiers sur les ports d'entrée et de sortie des acteurs sont le nombres de jetons respectivement consommés et produits par les acteurs à chaque exécution de ces derniers.

Une extension du modèle SDF est le modèle flux de données synchrone parametré et interfacé (PiSDF) [4] illustré dans la Figure 1. Dans le présent document, la caractéristique intéressante du modèle PiSDF est sa prise en charge de la hiérarchie. La hiérarchie permet de spécifier le comportement interne des acteurs par un sous-graphe au lieu d'un code C. Le modèle PiSDF définit les interfaces d'un sous-graphe comme des ports de données d'entrée et de sortie de l'acteur hiérarchique parent. Les interfaces permettent la transmission de jetons entre les niveaux hiérarchiques. La hiérarchie est utilisée pour représenter les différents niveaux de granularité des calculs qui composent une application, les niveaux inférieurs de la hiérarchie représentant la granularité la plus fine.

Ces modèles basés SDF présentent deux avantages principaux. Le premier est d'exprimer trois types de parallélisme [10], dont deux sont utilisés dans ce document : le parallélisme de tâches et le parallélisme de données (le pipeline étant le troisième). Le deuxième avantage est que le modèle est indépendant de l'architecture cible. Une application est représentée une seule fois et exécutable sur tous les types d'architecture (simple cœur, multicœurs avec mémoires partagées ou distribuées, FPGA, etc.)

2.2 Méthode d'aplatissement standard

Ce document se concentre sur les outils allouant des ressources au moment de la compilation, également appelé allocation statique. Le processus d'ordonnancement statique typique comprend quatre tâches principales, à savoir : aplatissement, transformation graphes acycliques dirigés à taux unique (SrDAG), placement et ordonnancement. La tâche d'*aplatissement* consiste à placer tous les acteurs d'un graphe au même niveau, ce qui signifie que tous les acteurs hiérarchiques sont remplacés par leur sous-graphe. La transformation SrDAG est utilisée pour révéler le parallélisme sur le graphe aplati. Elle met en évidence le nombre minimal d'exécutions de chaque acteur pour ramener le graphe à son état d'origine, donné par le calcul du vecteur de répétition vecteur de répétition (RV) **q**. Elle révèle également les interdépendances entre les acteurs, évitant ainsi les blocages lors de leur itération infinie. Chaque acteur du graph SrDAG est par la suite placé et ordonnancé individuellement. La complexité accrue du SrDAG augmente les possibilités de placement, ce qui permet de mieux répartir les calculs sur les différents PE et de réduire la latence de l'application sur la cible. Cependant, le placement est limité par le nombre de PE de l'architecture. Il s'avère donc inutile et chronophage d'exposer plus de parallélisme que le nombre de PE [6]. C'est pourquoi la réduction du parallélisme exposé au nombre de PE doit permettre d'exploiter pleinement le parallélisme de l'architecture tout en étant plus simple à placer et à ordonnancer.

2.3 Groupement d'acteur flux de données

Le regroupement des acteurs SDF est une technique efficace pour réduire la complexité des graphes. Étant donné que le regroupement de plusieurs acteurs en un seul acteur hiérarchique équivalent peut modifier le comportement de l'application ou même créer des blocages, des règles de regroupement d'acteurs ont été introduites dans [9] et illustrées sur quatre techniques de regroupement. La première est une méthode manuelle et fastidieuse qui donne à l'utilisateur la possibilité de sélectionner des groupes d'acteurs inappropriés introduisant des blocages. La deuxième consiste à regrouper des acteurs partageant les mêmes ressources sans introduire de blocages. La troisième est la méthode nommée nombre de répétitions uniques (URC). Cette méthode consiste à créer un sous-graphe SDF d'au moins deux acteurs séquentiels ayant un RV **q** identique et aucun état interne. La dernière a trait à l'allocation dynamique des ressources entrainant une surchage de travail sur les PE. Une autre technique de regroupement décrite dans [3], la méthode nommée regroupement par paire des nœuds adjacents (PGAN), consiste à coupler des paires d'acteurs dans le graphe de manière itérative, jusqu'à ce que l'application ne soit plus composé que d'un unique acteur, dont le contenu représente un ordonnancement monocoeur de tous les acteurs groupés. Cette méthode offre un large choix de configurations possibles dont l'évaluation est fastidieuse. Son extension, la méthode PGAN pour un graphe acyclique (APGAN) introduite dans [2] montre que coupler en premier les acteurs avec des poids de FIFO les plus élevés réduit l'empreinte mémoire et minimisent les configurations possibles. Toutes ces méthodes réduisent la complexité des graphes en augmentant la granularité de la description sans tenir compte des contraintes d'architecture en sacrifiant tout le parallélisme.

3 Méthode SCAPE

3.1 Optimisation de l'exploration de l'espace de conception

La méthode de regroupement proposée réduit la complexité du graphe tout en considérant l'architecture et intervient juste avant le processus d'aplatissement. La méthode SCAPE se compose de trois étapes : configuration de la granularité, identification de motifs particuliers qui feront l'objet d'un regroupement, et mise à l'échelle des derniers groupements sur l'architecture cible.

3.1.1 Configuration de la granularité

La méthode SCAPE reçoit en entrée un paramètre n_c qui correspond au nombre de niveaux de hiérarchie à regrouper grossièrement. Quand les n_c plus bas niveaux de hierarchie sont groupés, la méthode identifie deux motifs particuliers d'acteurs sur le niveau le plus bas du graphe transformé. Ces motifs sont décrits dans la Section 3.1.2. Ensuite les acteurs en question sont remplacés par un acteur hiérarchique modifié dont le contenu est un sous-graphe composé des acteurs décrit dans la Section 3.1.3. Puis la méthode génère un code associé au sous-graphe généré. Enfin le comportement de l'acteur hiérarchique est remplacé par ce code décrit dans la Section 3.2. De ce fait il existe n configurations de groupement possible, où n correspond au nombre de niveaux hiérarchiques du graphe d'entrée, auxquels s'ajoutent deux autres niveaux. Le niveau 0 correspond à la configuration de graphe sans groupement et le niveau $n+2$ au groupement de l'entièreté du graphe dans un seul acteur.

3.1.2 Identification de motifs particuliers

La méthode SCAPE prend en compte deux modèles : Le motif URC introduit dans la section 2.3 et le motif vecteur de répétition isolé (SRV). Ce dernier correpond à un acteur unique qui ne fait pas partie des candidats URC, avec un RV $\mathbf{q}$ supérieur ou égal au nombre de PE de l'architecture cible. Il s'agit donc d'un acteur dont le nombre d'éxécutions parallélisables dépasse le nombre de PE de l'architecture.

Exemple. On considère le graphe G de la Figure 1 et une architecture de 4 PE. G contient une séquence d'acteur D, E dans le sous-graphe h avec un RV $\mathbf{q}(D) = \mathbf{q}(E)$ connecté l'un l'autre par une FIFO sans délai. Les acteurs D, E sont des candidats URC. L'acteur F du graphe G est quant à lui un acteur isolé avec un RV $\mathbf{q}(F) = 8 > 4PE$. F est donc un candidat SRV.

3.1.3 Mise à l'échelle des groupements

La deuxième étape de la méthode consiste à générer un sous-graphe contenant les groupements d'acteurs préalablement identifiés. Le RV de l'acteur hiérarchique $\mathbf{q}(h_a)$ parent du sous-graphe est par la suite modifié pour correspondre à l'architecture cible, ce processus est appelé *mise à l'échelle*. Selon [7] pour préserver la cohérence du graphe G, sur chaque FIFO f les taux de jetons consommés et produits $cons$ et $prod$ et la RV $\mathbf{q}$ des acteurs source et destination src et dst sont liés par l'équation :

$$\mathbf{q}(src(f)) \times prod(f) = \mathbf{q}(dst(f)) \times cons(f) \qquad (1)$$

Pour calculer le facteur d'échelle, le RV de l'acteur hiérarchique $\mathbf{q}(h_a)$ doit être égal au diviseur commun des RV des acteurs du sous-graphe aplati juste au-dessus du nombre de PE. Dans le cas où l'acteur hiérarchique est connecté ou contient une FIFO avec un certain nombre de délais, une attention particulière doit être apportée lors de la mise à l'échelle de l'acteur. En effet, le facteur d'échelle est indexé sur la valeur du délai de sorte que le taux de jetons consommés sur la FIFO retardée soit inférieur ou égal à la valeur du délai. Pour maintenir la cohérence du graphe, les acteurs du sous-graphe sont exécutés $\mathbf{q}(a \in C)/\mathbf{q}(h_a)$ fois.

Exemple. On considère les candidats URC et SRV de l'exemple précédent. Les RV des acteurs identifiés D, E *et* F sont tous initialement égaux à 8. Si on considère une architecture de 4 PE alors le RV $\mathbf{q}(urc) = \mathbf{q}(srv) = 4$ et leur contenu $\mathbf{q}(D) = \mathbf{q}(E) = \mathbf{q}(F) = 2$. On se retrouve alors avec un SrDAG de 12 acteurs au lieu de 28 sans groupement d'acteurs avec un parallelisme préservé sur 4 PE.

3.2 Génération de code

Le code généré par l'approche standard d'aplatissement dans notre outil [8] prend la forme d'un fichier C spécifique pour chaque PE cible. Chaque fichier contient tout d'abord une partie dédiée à l'initialisation de l'application qui comprend la définition des tampons alloués, des acteurs et des FIFO des fonctions d'initialisation telles que l'initialisation du délai. La deuxième partie de ces fichiers est une boucle représentant le thread contenant le déclenchement programmé des acteurs. Il s'agit d'un appel de fonction mettant en œuvre le comportement de l'acteur. La méthode proposée génère du code spécifique au groupement d'acteur et prend également la forme d'un fichier C. Un groupement d'acteurs est traduit par des appels de fonctions imbriqués selon si le groupe contient ou non d'autres groupes et le RV des éléments du groupe est traduit par des boucles "for".

4 Expériences

4.1 Mise en place des expériences

La méthode SCAPE est appliquée sur trois applications de traitement d'image : Stereo, Stabilization et Squeezenet sur une description PiSDF résumé dans le Tableau 1. Ces applications ont entre 2 et 3 niveaux de hiérarchie et présentent peu de délais dans leur description.

Étant donné que la méthode proposée fournit un ensemble de configurations de groupement, celle qui offre le meilleur compromis en termes de nombre d'acteurs SrDAG, de temps de processus d'allocation des ressources, également appelé temps d'analyse, et de latence est choisie.

Du fait que les critères de performance dépendent de l'architecture et que la méthode exploite ces informations dans la transformation, les expériences ont été menées sur des architectures avec 1, 2, 4, 8 ou 16 cœurs homogènes.

La méthode a été implémentée dans des projets open source dans l'environnement de prototypage rapide méthode d'ordonnancement des exécutifs embarqués en temps réel et en parallèle (PREESM). Les expériences sont réalisées sur un ordinateur avec un processeur Intel i7-8665U à 8 cœurs et 31,2 Go de RAM.

TABLE 1 : Comparaison du temps d'analyse et de latence entre l'approche sans groupement de l'état de l'art et la meilleure configuration de groupement de la méthode SCAPE sur trois cas d'utilisation, *Valeurs estimées

					Nombre de PE				
Application	**SDF**	**Niveau**	**SrDAG**	**Temps relatif**	1	2	4	8	16
Stereo	28	2	187	analyse	5.3	1.2	1.4	1.7	1.6
				latence	1.0	0.9	0.9	0.8	0.8
Stabilization	22	3	98	analyse	1.5	0.5	0.5	0.6	0.7
				latence	1.0	1.0	0.7	0.7	0.8
Squeezenet	98	3	5452	analyse*	203.5k	100.5	94.5	84.2	68.8
				latence*	1.0	1.0	1.0	1.0	1.0

4.2 Evaluation du temps d'analyse et de la latence

Les résultats présentés dans le Tableau 1 correspondent au rapport entre les temps obtenus sur la configuration sans groupement, « niveau 0 », et ceux obtenus sur la configuration de groupement offrant le meilleur compromis entre les temps d'analyse et de latence. Une valeur supérieure à 1 est une accélération de la méthode. Concernant le temps d'analyse, la plupart des valeurs sont des accélérations grâce à la méthode sauf pour les très petites applications telle que Stabilization. En effet la méthode réduit la complexité d'un graphe déjà très simple, ainsi les gains de temps d'allocation des ressources sont contrebalancés par le temps de procédure de groupement d'acteurs. A contrario, les gains de temps d'analyse sont important sur les graphes très complexes tel que Squeezenet. Les ratios sur cette application sont estimés pour la configuration sans groupement puisque l'analyse d'un graphe si complexe dépasse les capacités de RAM de la machine utilisée et n'a pas pu se terminer après 48h. D'où la pertinence de la méthode de fournir des analyses et du code exécutable même sur des applications très complexes. Concernant le temps de latence du code généré, la méthode dégrade légèrement la sortie à cause du ré-ordonnancement des acteurs induit par le regroupement d'acteurs. Cette dégradation devient négligeable pour des graphes d'entrée complexes tel que Squeezenet.

5 Conclusion

Cet article présente une nouvelle méthode pour réduire le temps de placement et d'ordonnancement tout en préservant le parallélisme des graphes SDF. Elle consiste à réduire la taille du graphe en regroupant des acteurs reproduisant des motifs particuliers puis à adapter le parallelisme des groupements à l'architecture cible. La méthode permet à l'utilisateur de choisir l'expansion potentielle du programme produit et de réduire le temps d'analyse en conséquence. Les résultats expérimentaux montrent que pour un temps d'analyse significativement amélioré, on obtient une latence légèrement dégradée du code généré. De plus, la méthodes permet de placer et d'ordonnancer des applications massivement parallèles qui étaient trop complexes pour les approches de l'état de l'art. Les orientations potentielles pour les travaux futurs incluent l'identification et le regroupement de modèles plus complexes et l'automatisation de la recherche du niveau optimal de regroupement, sans avoir à essayer toutes les configurations.

6 Bibliographie

Références

[1] G. ALOK : Architecture apocalypse dream architecture for deep learning inference and compute-versal ai core. *Embedded World*, 2020.

[2] S. BHATTACHARYYA et AL. : Apgan and rpmc : Complementary heuristics for translating dsp block diagrams into efficient software implementations. *Design Automation for Embedded Systems*, 1997.

[3] S. S. BHATTACHARYYA et E. A. LEE : Scheduling synchronous dataflow graphs for efficient looping. *Journal of VLSI signal processing systems for signal, image and video technology*, 1993.

[4] K. DESNOS et AL. : Pimm : Parameterized and interfaced dataflow meta-model for mpsocs runtime reconfiguration. *2013 International Conference on Embedded Computer Systems : Architectures, Modeling, and Simulation (SAMOS)*, 2013.

[5] E.C. KLIKPO et AL. : Modeling multi-periodic simulink systems by synchronous dataflow graphs. *In 2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2016.

[6] E.A. LEE et S. HA : Scheduling strategies for multiprocessor real-time dsp. *In 1989 IEEE Global Telecommunications Conference and Exhibition 'Communications Technology for the 1990s and Beyond'*, 1989.

[7] E.A. LEE et D.G. MESSERSCHMITT : Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Transactions on Computers*, 1987.

[8] M. PELCAT et AL. : Preesm : A dataflow-based rapid prototyping framework for simplifying multicore dsp programming. *In 2014 6th European Embedded Design in Education and Research Conference (EDERC)*, 2014.

[9] J.L. PINO et AL. : A hierarchical multiprocessor scheduling system for dsp applications. *In Conference Record of The Twenty-Ninth Asilomar Conference on Signals, Systems and Computers*, 1995.

[10] Z. ZHOU et AL. : Scheduling of parallelized synchronous dataflow actors. *In 2013 International Symposium on System-on-Chip, SoC 2013 - Proceedings*, 2013.