Méthodologie de synthèse de très haut niveau pour des architectures hétérogènes logicielles et matérielles

Gaëtan Lounes, Robin Gerzaguet, Matthieu Gautier

Méthodologie de synthèse de très haut niveau pour des architectures hétérogènes logicielles et matérielles

Gaëtan Lounes, Robin Gerzaguet, Matthieu Gautier
Univ Rennes, CNRS, IRISA
prenom.nom@irisa.fr

***Abstract*—L'utilisation croissante des circuits logiques programmables (Field Programmable Gate Arrays, ou FPGA) a permis l'accélération matérielle d'un large éventail d'applications. L'introduction de la synthèse de haut niveau (High Level Synthesis, ou HLS) a considérablement simplifié leur conception et permet l'exploration d'applications complexes. Cependant, la HLS demande encore une solide connaissance du matériel ciblé et repose sur des langages spécifiques. De nouveaux outils HLS s'appuie ainsi sur des environnements de compilation plus expressifs tels que la représentation intermédiaire à plusieurs niveaux (Multi-Level Intermediate Representation, ou MLIR). Dans ce contexte, nous visons à utiliser le langage de programmation Julia comme une interface flexible pour une HLS basée sur MLIR, en tirant parti de son système de types, de son infrastructure modulaire et de ses bibliothèques riches. Dans cet article, nous introduisons un nouveau compilateur qui exploite l'infrastructure de compilation de Julia afin de générer du code MLIR, facilitant ainsi l'accélération matérielle. L'intérêt de cette approche a été démontré pour du prototypage rapide et de l'exploration des espaces de types pour des algorithmes numériques.**

***Index Terms*—Synthèse de haut-niveau, Langage Julia, MLIR**

I. Introduction

Les Field Programmable Gate Arrays (FPGA) sont une classe de matériel configurable qui a démontré son efficacité pour des applications nécessitant une faible latence et un haut débit, telles que le traitement du signal ou les réseaux de neurones. Ils sont configurés à l'aide de langages de description matérielle (HDL) comme le VHDL. Toutefois, cette approche n'est pas idéale pour le développement rapide d'algorithmes ni pour l'exploration de solutions. La synthèse de haut niveau (High Level Synthesis, HLS), comme par exemple l'outil Vitis, comble partiellement cet écart en traduisant une description comportementale écrite dans un langage spécifique, proche des paradigmes de programmation classiques, vers une sémantique HDL.

Le langage de programmation Julia [1] comble un besoin similaire du côté logiciel en simplifiant la conception : Julia vise à produire directement des programmes très performants à partir d'un langage de haut niveau flexible. Pour atteindre cet objectif, Julia repose sur plusieurs principes : un système de types expressif, une typage progressif (on parle d'inférence), et la métaprogrammation. En particulier, Julia utilise une représentation intermédiaire (IR) qui est générée et transformée durant la compilation, avant d'être envoyée à la chaîne d'outils LLVM pour produire du code machine (CPU). Ces principes ont déjà été utilisés pour développer des back-ends vers d'autres cibles matérielles comme les GPU ou les TPU, mais les matériels spécialisés comme les FPGA n'ont pas encore été explorés.

Cet article présente un nouveau *back-end* Julia exploitant ces principes pour l'accélération matérielle, en utilisant une nouvelle forme de représentation intermédiaire issue de l'environnement de compilation Multi-Level Intermediate Representation (MLIR) [2]. MLIR est un framework qui généralise le concept d'IR en permettant la création de plusieurs sémantiques IR avec différents niveaux d'abstraction. Ce framework est notamment utilisé pour décrire des chaînes de compilation complexes, comme ScaleHLS [3], qui exploite l'expressivité et la modularité de MLIR pour créer un cadre HLS capable d'annoter automatiquement du matériel. Cet article propose donc une chaîne HLS complète allant du langage Julia jusqu'à une description HDL, en s'appuyant sur MLIR et ScaleHLS, permettant ainsi un prototypage rapide et des formes avancées d'exploration de l'espace de conception (Design Space Exploration, DSE).

La section 2 présente la chaîne de compilation de Julia vers MLIR et ScaleHLS. Ensuite, la section 3 présente un cas d'usage de ce nouveau compilateur. Enfin, la section 4 conclut cette étude.

II. Judias : introduction de MLIR et ScaleHLS dans le flot de compilation Julia

La chaîne de compilation Julia repose sur l'utilisation de LLVM [4] : l'intérêt de cette machine virtuelle est avant tout de permettre la génération de code machine. Ainsi, une partie de la sémantique du programme source est perdue, ce qui est préjudiciable car les annotations de Vitis ne peuvent pas être générées à partir du LLVM IR obtenu de Julia [5]. MLIR se distingue par sa capacité à représenter des programmes avec différents niveaux d'abstraction, ce qui le rend particulièrement intéressant pour modifier et optimiser le code à différentes échelles. Une des notions clés de MLIR est l'utilisation de "dialectes", qui permettent de définir des opérations spécifiques à un domaine ou à une architecture matérielle. Cet écosystème a démontré son utilité dans les

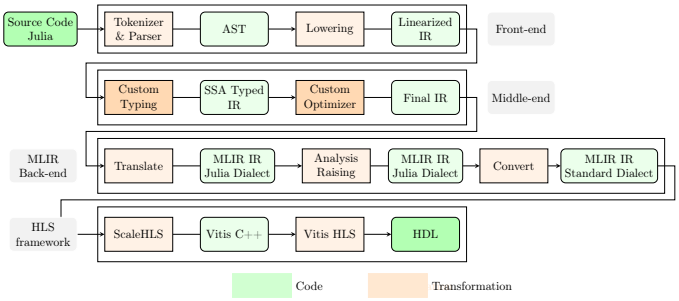

Fig. 1. Judias : Compilateur Julia modifié générant du MLIR IR.

compilateurs d'apprentissage machine ou dans le cadre de la HLS.

Par exemple, ScaleHLS [3] et son successeur HIDA sont des frameworks HLS qui utilisent MLIR pour définir le dialecte du matériel et permettre la génération de programmes HLS performants. Pour atteindre cet objectif, un pipeline de compilateur composé de plusieurs niveaux d'abstraction a été introduit, en commençant par les dialectes standard MLIR et en terminant par un dialecte Vitis personnalisé qui est traduit en Vitis HLS C++ synthétisable.

L'idée centrale de l'approche proposée dans cette étude, illustrée dans la Figure 1, est d'introduire la génération de MLIR juste avant l'étape de traduction vers LLVM, afin de maximiser les avantages offerts par le compilateur Julia. Cette stratégie permet de préserver le système *multiple dispatch* [4] et le système de types de Julia tout en exploitant des optimisations avancées. Seul un sous-ensemble du langage Julia est pris en charge : les programmes doivent être entièrement typés, sans appel de fonctions externes ni utilisation du runtime Julia, comme la réflexion. Ainsi, l'IR de Julia est parcourue et transformée en code MLIR, avec certaines constructions spécifiques, telles que les *phinodes*, converties en blocs fonctionnels MLIR.

Afin de faciliter l'intégration avec les dialectes MLIR standards, un dialecte MLIR personnalisé pour Julia est créé. L'objectif de ce dialecte est de simplifier la traduction, notamment pour la conversion de types et la passe d'élévation (*rising*). Ensuite, l'étape MLIR se déroule en deux phases :

- Une passe de *rising* : l'IR de Julia, étant non structuré, les boucles for et les structures if sont perdus durant la compilation. Ces structures sont extraites en tirant parti de la spécificité des itérateurs de Julia et de l'analyse de dominance.
- Plusieurs passes de *lowering* : les opérations et types Julia restants du dialecte personnalisé sont convertis en leurs équivalents MLIR, notamment dans les dialectes standards tels que `func`, `arith`, `index`, `memref` et Structured Control Flow (SCF). Cette étape est détaillée plus en profondeur dans [6].

III. Analyse comparative sur GEMM

Pour démontrer les capacités de ce compilateur à générer du code MLIR utilisable pour ScaleHLS, nous considérons l'exemple GEMM, une multiplication de matrices 3x3 en Float32. L'objectif est de démontrer qu'à partir de la même base de code flexible, nous pouvons générer des architectures matérielles. Nous proposons ici de comparer les résultats du MLIR généré avec le code C fourni par Polygeist [7]. Les résultats en ressources utilisées et nombres de cycles sont donnés dans la Table I.

TABLE I
Comparaison des métriques entre Polygeist et notre approche.

Métriques	Polygeist	De Julia
Cycles	31	35
DSP	34	34
FF	4195	4037
LUT	3114	3032

Les résultats montrent que les métriques sont très similaires. La solution générée par Polygeist présente un nombre légèrement inférieur de cycles d'horloge mais nécessite davantage de ressources matérielles. Le point clé ici est que la signature Julia est paramétrique, tandis que le code C fourni par Polygeist est typé statiquement. Cette flexibilité de Julia facilite une exploration plus polyvalente de l'espace de conception.

IV. Conclusions

Dans cet article, nous présentons un *back-end* Julia MLIR adapté à la synthèse de haut niveau, réalisé grâce à une pipeline de compilateur modulaire qui effectue une conversion statique d'IR vers IR sur des programmes Julia entièrement typés. Cette nouvelle approche est comparée à Polygeist à l'aide d'un benchmark de référence et permet une flexibilité dans l'exploration de l'espace de types.

References

[1] T. Besard, C. Foket, and B. De Sutter, "Effective Extensible Programming: Unleashing Julia on GPUs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 4, pp. 827–841, Apr. 2019.

[2] C. Lattner, M. Amini, U. Bondhugula, A. Cohen, A. Davis, J. Pienaar, R. Riddle, T. Shpeisman, N. Vasilache, and O. Zinenko, "MLIR: A Compiler Infrastructure for the End of Moore's Law," Feb. 2020.

[3] H. Ye, C. Hao, J. Cheng, H. Jeong, J. Huang, S. Neuendorffer, and D. Chen, "ScaleHLS: A New Scalable High-Level Synthesis Framework on Multi-Level Intermediate Representation," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, Seoul, Korea, Republic of, Apr. 2022, pp. 741–755.

[4] J. Bezanson, J. Chen, B. Chung, S. Karpinski, V. Shah, J. Vitek, and L. Zoubritzky, "Julia: Dynamism and performance reconciled by design," vol. 2, pp. 1–23.

[5] G. Lounes, R. Gerzaguet, and M. Gautier, "Julia meets the FPGA : Higher-level synthesis methodology for heterogeneous hardware and software architectures," Conference on the Julia programming language (JuliaCon), p. 1, Jul. 2024.

[6] ——, "Flexible front-end for high level synthesis leveraging heterogeneous compilation," in *Workshop on Rapid Simulation and Performance Evaluation for Design Optimization: Methods and Tools (RAPIDO)*, Barcelona (ES), Spain, Jan. 2025.

[7] W. S. Moses, L. Chelini, R. Zhao, and O. Zinenko, "Polygeist: Raising c to polyhedral mlir," in *Proc. 30th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2021, pp. 45–59.