Enabling Flexible Model-Based Manually Controllable Compilation for CGRAs

PHD FORUM SUBMISSION

Felix Böseler
OFFIS - Institute for Information Technology
Oldenburg, Germany
felix.boeseler@offis.de

I. Introduction

The "power wall" is a well known problem in modern microprocessor design [1]. Accelerators can be used to offload computation-intensive kernels encountered in a wide range of digital signal processing applications. An interesting accelerator type is the so-called Coarse-Grained Reconfigurable Array (CGRA). Through the parallel computation capabilities on multiple granularities paired with partial specialization, a CGRA can achieve very high energy efficiency [1].

In industrial development processes, experts from different domains (e.g., control theory or hardware) often want to have full control over the entire compilation process. For CGRAs, this means full control beginning from the mathematical kernel description down to the CGRA configuration bitstream. Experts often have to manually guide and explore optimization and lowering steps to achieve good CGRA utilization [2]. Furthermore, CGRAs may be used in safety-critical applications such as automotive engine control units (see [3]) where full control over the compilation process increases traceability.

Therefore, the goal of this thesis is to enable a flexible model-based manually controllable compilation flow for CGRAs as seen in Fig. 1. It shows that an external kernel description (e.g., procedural description) is converted through a parsing or transformation step into a high-level model. This high-level model is described in a flexible model-based language, which allows to gradually optimize and lower the high-level model through multiple expert-driven transformation rules into a low-level model. The low-level model includes technology mapping and is therefore in a form that enables a Placing & Routing (P&R) step to assign concrete CGRA resources to every model element. Finally, this *placed* model is encoded into a configuration bitstream.

While related work such as the LLVM MLIR framework [4] is very flexible, it is not designed for human readability. Conversely, numerous compilation flows exists that are understandable but focus on low-level abstractions or fixed CGRA architectures (see [2]). The proposed flow is novel in that it selects a careful balance between the understandability and flexibility requirements to increase CGRA compilation accessibility. Ensuring understandability allows full control for domain experts so that transient partially lowered models can be inspected. Allowing flexibility helps to decrease the already existing fragmentation of CGRA compilers.

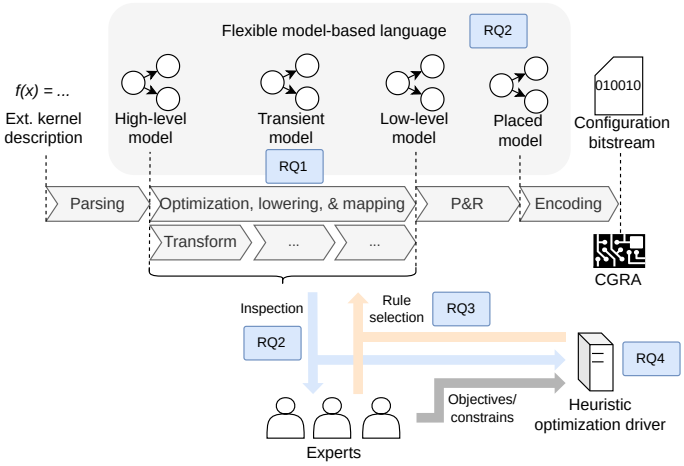

Fig. 1. Flexible model-based manually controllable compilation flow

Therefore, the main research question of this thesis is as follows: how can a flexible model-based manually controllable compilation flow from a mathematical kernel description to a CGRA configuration be realized? To answer this main research question, the next section details further research questions.

II. Research Questions

Fig. 1 shows the relation between the proposed compilation flow and the research questions through blue rectangles. It shows that RQ1 is concerned with the definition of the necessary abstraction levels between the high-level model and the low-level model. RQ2 examines how all transient models between these abstraction levels can be described in a flexible modeling language. However, it is crucial that this flexibility does not hinder the understandability for experts.

The goal of RQ3 is to answer the question how an expert can describe model transformations in an understandable way. This question is closely related to the reusability of components (see RQ3.1). Finally, RQ4 is concerned with the task how heuristics, such as simulated annealing, can be integrated in the compilation task. When applying such a heuristic optimization, the experts should have the possibility to integrate own objective preferences or constraints so that the manual controllability can still be ensured.

The detailed descriptions of the research questions are given below:

- RQ1 What levels of abstraction are necessary to facilitate understandability so that manual controllability in the compilation flow can be achieved?
- RQ2 Based on these levels of abstractions, how does a flexible model-based graphical language look that can describe transient optimization, lowering, and technology mapping models?
- RQ2.1 What is a useful execution semantic for such a language?
- RQ3 How should the model-to-model transformation flow for optimizaton, lowering, and mapping look like?
- RQ3.1 How can the model-based language support reusable generic components?
- RQ3.2 What transformation rule format enables understandable transformations of suitable capability?
- RQ4 How can heuristics, such as simulated annealing, be integrated into the compilation flow so that more complex optimization, lowering, and mapping steps can be realized.

III. Current State

My current work regarding RQ1 and RQ2 is described in [5] where a hierarchic Data Flow Graph (DFG) language comparable to approaches such as [6] is proposed. However, in contrast to [6], [5] provides the important possibility to abstract from shared memory details for multidimensional data in higher abstractions. [5] argues that the focus on kernels with constant loop boundaries is sufficient for an initial DFG language version. Furthermore, the paper briefly reviews CGRA taxonomies and identifies the class of heterogeneous, statically reconfigurable, and dynamically scheduled CGRAs (see [7]) as sensible CGRA target type for the proposed compilation flow. This type of CGRA is called dataflow CGRA in [5].

My current work regarding transformation rules (see RQ3) focuses on declarative pushout transformation rules as already often found in model-driven development contexts (see [8]). While first examinations suggest that such declarative rules are well suited for lowering transformations, they may be less suited for general optimization steps (e.g., loop unrolling). A difficult decision is whether declarative rewriting rule extensions such as multi-objects (see [8]) should be supported or whether procedural rules should be employed as fallback for such cases (see also [4]). This question is especially interesting in the context of the understandability requirement of the proposed compilation flow, since declarative rules in their full expressiveness may offer decreased readibility.

IV. Evaluation

The planned overall evaluation relies on case studies that should evaluate the two main requirements of the overall compilation flow – understandability and flexibility.

While I provide first conceptual compilation results in [5], extended quantitative evaluations are required once the progress of the thesis has further advanced. I plan to continue focusing on the DFA CGRA (see [3]) because of its industrial relevance. While this places an external dependency on the thesis, the project partners have expressed explicit motivation to continue collaboration beyond the current project duration. Furthermore, hardware simulation possibilities for the targeted CGRA exist.

To provide quantitative comparable results regarding understandability, interesting metrics are the required end-to-end time for kernel compilation and CGRA utilization/performance of produced CGRA configurations. The industrial partners can provide a rich set of industrial relevant kernels such as fast Fourier transformation, Gaussian process, and multi layer perceptron. The CGRA configurations for these kernels are currently mostly handcrafted individually. I think that the proposed compilation flow can provide a measurable advantage compared to the current handcrafted configuration approach.

The flexibility should be evaluated by focusing at least one alternative CGRA. I identify as a main challenge to find a contemporary dataflow CGRA with industrial relevance for which also accessible hardware simulation possibilities exist.

Naturally, the developed compiler in this thesis has to be compared to existing CGRA compilers. However, this proves to be difficult and a main challenge of this thesis because of the vast and rapidly evolving amount of CGRA compiler frameworks. New CGRA compiler/architecture exploration frameworks can be observed almost every year. Thus, it is crucial to filter the existing set of compiler approaches and only focus on the most similar approach for a direct comparison. I would be especially happy to discuss ideas to handle this tasks or to potentially alleviate this task through adapted research questions.

References

[1] L. Liu et. al, "A survey of coarse-grained reconfigurable architecture and design: Taxonomy, challenges, and applications," *ACM Comput. Surv.*, vol. 52, no. 6, pp. 1–39, Oct. 2019, doi: 10.1145/3357375.

[2] J. M. P. Cardoso, P. C. Diniz, and M. Weinhardt, "Compiling for reconfigurable computing: A survey," *ACM Comput. Surv.*, vol. 42, no. 4, Jun. 2010, doi: 10.1145/1749603.1749604.

[3] J. Froemmer, N. Bannow, A. Aue, C. Grimm, and K. Schneider, "Flexible data flow architecture for embedded hardware accelerators," in *International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP) 2019, Part I*, 2020, pp. 33–47, doi: 10.1007/978-3-030-38991-8_3.

[4] C. Lattner et al., "MLIR: Scaling compiler infrastructure for domain specific computation," in *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, 2021, pp. 2–14, doi: 10.1109/CGO51591.2021.9370308.

[5] F. Böseler and J. Walter, "A flexible graph language for a model-based semi-automatic CGRA compilation flow," in *Forum on specification & Design Languages (FDL) 2023*, Sep. 2023.

[6] G. Suba, "Hierarchical pipelining of nested loops in high-level synthesis," *Periodica Polytechnica Electrical Engineering and Computer Science*, vol. 58 (3), p. 81–91, Jan. 2014, doi: 10.3311/PPee.7610.

[7] B. De Sutter, P. Raghavan, and A. Lambrechts, "Coarse-grained reconfigurable array architectures," in *Handbook of Signal Processing Systems*, S. S. Bhattacharyya, E. F. Deprettere, R. Leupers, and J. Takala, Eds. New York, NY: Springer New York, 2013, pp. 553–592.

[8] R. Heckel, "Graph transformation in a nutshell," *Electronic Notes in Theoretical Computer Science*, vol. 148, no. 1, pp. 187–198, 2006, doi: 10.1016/j.entcs.2005.12.018.