Modelling and Mapping Framework for Coarse-Grained Programmable Architectures

E. Barbudo[1] E. Dokladalova[1] Th. Grandpierre[1]
[1] LIGM, Université Gustave Eiffel, UMR CNRS 8049, ESIEE Paris,
F-77454 Marne-la-Vallée, France
{surname.lastname}@esiee.fr

1 Introduction

Coarse-Grained Reconfigurable Architectures (CGRA) deliver high performance with low latency. They are highly domain-specifically optimized with some defined level of flexibility [1, 2]. We consider a CGRA as programmable configuration-driven computing fabric with fixed post-fabrication flexibility. Hence, in this work, we use the term "programmable" instead of "reconfigurable" architecture.

A Coarse-Grained Programmable Architecture (CGPA) consists of an irregular array of heterogeneous processing, communication, and memory resources, together with a configuration support layer [3]. A CGPA may consist of several parallel pipelines with varying lengths. This feature exhibits different levels of parallelism and allows one to execute concurrent tasks in an asynchronous parallel mode. The processing resources are heterogeneous modules, with possible different computing models designed to perform a specific set of tasks. Each processing resource may have a set of programmable parameters. The combination of different computing models (e.g. image processing filters) with their parameters may produce an individual latency for each processing resource. The communication resources allow controlling data-paths between processing resources. Finally, the memory resources are used to store either input, partially processed data to be re-used in the next step/iteration, or output data.

Considering the heterogeneous computing models combined with the cost of memorization, communication, and configuration, application mapping to a CGPA is not trivial.

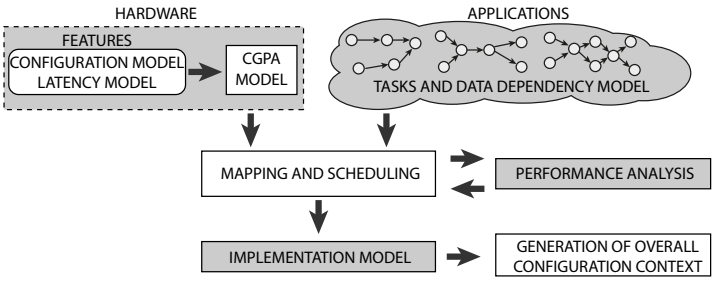

Figure 1: Overall scheme of our general modelling and mapping framework for CGPA.

Our proposed solution is a modelling and mapping framework (Figure 1). The core of this solution must be a modelling tool allowing one to describe the heterogeneity of the computing models and physical realizations of any CGPA hardware element. In addition to the hardware model, the tool must integrate corresponding models of application and implementation. Also, the framework requires a mapping and scheduling algorithm that takes as inputs the application and hardware models and provides the implementation model as output. This algorithm should work together with

a performance analysis that estimates the latency timings. Finally, the produced implementation model should contain information on the configuration context.

In the literature, there are numerous automated application mapping methodologies: REmulator [4], CGRA-ME [5], DRESC [6]. These tools are custom for a specific platform, or they generate a platform according to the application. Also, at the model level, they do not provide the means to accurately model the latencies of the hardware resources or neglect elements of the CGPA [7].

In our work, we introduce a new graph-based modelling tool (Section 2). We model the application and the hardware as directed hypergraphs. Lastly, we model the implementation as a weighted directed hypergraph.

We propose two mapping algorithms (Section 3). First, the exhaustive mapping algorithm is able to get an optimal mapping, by exploring all possible mappings and selecting the one with the lowest computing cost. We define the computing cost as the time elapsed from the arrival of the first sample until the output of the last processed sample, including also the configuration cost. Obviously, this exhaustive algorithm is time-consuming. The second algorithm, based on a list scheduling heuristic, is able to get a mapping in a significantly shorter time, in order of tens of seconds.

To estimate the timings of the computed mapping, we propose a performance analysis (Section 4) with the computing cost of the implemented application as the metric (computed in clock cycles). Finally, in Section 5 we summarize this abstract and outline the perspectives of this work.

2 Modelling

Our framework is based on three graph-based models. G_{APP} specifies the task dependency and application parameters. G_{HW} describes the hardware resources and their interconnections. We divide the hardware resources (nodes) of G_{HW} into configuration control, processing, communication, and memory resources. We provide, for each hardware resource, the means to model its latency functions, configuration cost function, and programmable parameters. Concerning the latency, we propose differentiate the **input latency** and **computing latency** of each hardware resource. We define the ***input latency*** as the number of clock cycles needed to read all the samples required to start to compute the first result. We define the ***computing latency*** as the number of clock cycles necessary to produce the result once all input samples are available.

By matching G_{APP} onto G_{HW}, we produce a new graph G_{MAP} using mapping and scheduling presented in Section 3. Each node of G_{MAP} has parameters fixed during the mapping and the weight of the edges represents the input latency of the head node. G_{MAP} consists of one or more **time slots** created on demand. A time slot is a subset of resources configured to perform a subset of tasks. Hence, adding a time slot allows re-using the hardware which can be reconfigured between time slots when resources are missing.

3 Mapping and scheduling

The exhaustive algorithm is able to get the optimal mapping by building iteratively all the possible mappings and providing the computing cost estimation. We start with the topological sorting of the application graph. We test the possibility to map a task to any available hardware resource that complies with the requirements. We consider the possibility to map one task in one time slot. The idea behind this is that the physical realization of one hardware resource may be efficient enough to compensate for the configuration cost of each time slot generated. As a result of this consideration, we produce an optimal mapping, but it takes a very long time to be computed.

We base our heuristic algorithm on look-ahead techniques [8]. This approach not only tries to find an allocation for a task but also considers the mapping of the task's successors. We start with a topological sorting of the application graph and a list with the source nodes of the hardware graph. We select the first task $t_i \in G_{APP}$ and try to map it to any source node of the hardware graph. For this purpose, we get the successors of t_i and the descendants of the source node $r_j \in G_{HW}$. We compute the feasibility of allocating the successors of task t_i onto the descendants, taking into account the topological distance. We perform the same computation to all the source nodes and we select the source node with the highest feasibility value.

We are conducting several tests on real and random generated hardware examples to demonstrate that the near-optimal mapping is produced. The exploration cost is significantly lower than exhaustive exploration, which makes this algorithm suitable for run-time scheduling.

4 Performance analysis

The performance analysis is based on the computing cost. We propose a method that computes the input and computing latency of all the elements of the critical path. It is able to consider different latency functions for each hardware resource and propagate these different values through the critical path. We propose to integrate the configuration cost, often neglected in other works.

5 Conclusions

We have introduced our proposed modelling and mapping framework. Our models provide the means to abstract the configuration cost and the heterogeneous latency of a CGPA. We presented two mapping approaches. The exhaustive approach is able to produce an optimal mapping in terms of latency but at the cost of high exploration time. The look-ahead-based heuristic approach provides a mapping in less exploration time. Additionally, we define a method for performance analysis based on the computing cost over the critical path. We validated our framework with generated random systems with promising results. We are working on the application of our framework in a real-world case .

Acknowledgements.

This research is partially supported by the Mexican National Council for Science and Technology (CONACYT) under grant 471691.

References

[1] L. Liu et al., "A Survey of Coarse-Grained Reconfigurable Architecture and Design: Taxonomy, Challenges, and Applications," in *ACM Comput. Surv.*, Jan 2019, vol. 52.

[2] A. Chattopadhyay, "Ingredients of Adaptability: A Survey of Reconfigurable Processors," in *VLSI Des.*, Jan 2013.

[3] M. Wijtvliet et al., "Coarse grained reconfigurable architectures in the past 25 years: Overview and classification," in *IEEE 16Tth SAMOS*, July 2016, pp. 235–244.

[4] R. Chen et al., "Hierarchy modeling and co-simulation of a dynamically coarse-grained reconfigurable architecture," in *Informatics in Control, Automation and Robotics*, Ed. Springer Berlin Heidelberg, 2012, pp. 589–598.

[5] S. A. Chin et al., "Cgra-me: A unified framework for cgra modelling and exploration," in *IEEE 28th ASAP*, July 2017, pp. 184–189.

[6] B. Mei et al., "Dresc: a retargetable compiler for coarse-grained reconfigurable architectures," in *IEEE 2002 FPT*, Dec 2002, pp. 166–173.

[7] E. Barbudo et al., "A New Mapping Methodology for Coarse-Grained Programmable Systolic Architectures," in *22nd SCOPES*, May 2019, pp. 5–12.

[8] L. Bittencourt et al., "DAG Scheduling Using a Lookahead Variant of the Heterogeneous Earliest Finish Time Algorithm," in *18th PDP*, 2010, pp. 27–34.