A Mapping Methodology for Coarse-Grained Pipelined Configurable Architectures

Elias Barbudo, Eva Dokladalova, Thierry Grandpierre, Laurent George

A Mapping Methodology for Coarse-Grained Pipelined Configurable Architectures

E. Barbudo * E. Dokladalova * Th. Grandpierre * L. George *

1 Introduction

Coarse-grained pipelined configurable architectures allow us to meet hard time constraints and provide high-performance computing capabilities. Often, they consist of a scalable structure of heterogeneous resources with non-regular interconnections. Such architectures are optimized for a given applicative field. The mapping of new applications on these targets requires a broad understanding of the internal organization and configuration parameters, and it can become an overwhelming task. Therefore, the need for an automated application mapping methodology becomes critical.

A considerable number of approaches (graph-based [1], integer linear programming [2], simulated annealing [3]) brings numerous solutions for similar mapping problems. However, they focus mainly on homogeneous resources, and the application mapping for hardware architectures with limited heterogeneous resources and non-regular connectivity is still an open problem.

In this work, we present a new graph-based mapping methodology for coarse-grained pipelined configurable architectures. The proposed methodology uses a Directed Acyclic Graph (DAG) as task and hardware model. It performs a latency evaluation of the resulting mapping and provides a set of configuration parameters for the hardware.

2 Methodology

Our approach consists of three models described by a DAG formalism. G_{APP} specifies the task dependency and application parameters. G_{HW} describes the hardware resources and their directed interconnections. We describe each node of G_{HW} by the set of tasks that it can perform, the set of working parameters and the latency model. Finally, we obtain G_{MAP} by graph transformations between G_{APP} and G_{HW}. We describe each node of G_{MAP} by the properties assigned during the mapping process (task, parameters, latency).

*surname.lastname@esiee.fr. Université Paris-Est, LIGM UMR CNRS 8049, ESIEE Paris.

Three steps integrate the mapping algorithm. First, we perform a topological sorting of G_{APP} and G_{HW}. Second, the mapping algorithm searches for the correct match between tasks and resources. This matching respects the connectivity of the resources and the data dependency of the tasks. During this process, two problems may appear. The first problem, failure to match, refers to a repetitive unsuccessful search for a resource for a given task. To solve this issue, we use a modification of the backtracking algorithm presented by Lu et al. [1]. The last problem, unavailability of resources, refers to a condition created by the non-uniqueness of the topological sorting. To solve this, the algorithm verifies if there are any data-paths available. If the mapping algorithm is not able to find any available resources, it will proceed to cut G_{APP} into sub-graphs. Next, it will try to schedule them into time slots, which are a subset of configured resources designed to execute a subset of tasks. In order to execute the whole application, all time slots need to be executed sequentially. Finally, after a successful mapping, we obtain G_{MAP}. Using the critical path of G_{MAP}, we perform a latency evaluation. Moreover, we obtain the set of configuration parameters from parsing G_{MAP}.

3 Experimental results

We consider the Morphological Co-Processing Unit (MCPU) [4] as a candidate for the use of our mapping methodology and two different applications.

The first application is a long linear pipeline of tasks. This application exceeds the available resources. A naive way to implement this application is by several times slots, increasing the scheduling length. In contrast, an optimized form of implementation is through the use of all the available data paths of the hardware, but creating a data dependency between them.

The methodology was able to achieve the best use of resources and map the application in the least amount of time slots. The methodology solves the problem of data dependence by specifying the necessary configuration parameters for the correct processing of the data.

The second application, road line detection, represents a highly parallel task organization. The optimized way to implement the application is using all the available data-paths in the hardware. The mapping methodology was able to map correctly the application, achieving the lowest latency possible and the best use of resources. We obtain this as a result of the search for non-used resources of the mapping algorithm, that allow us to find the unused data-paths.

For both applications, the results were equal to manual mapping. In addition, the latency evaluation provides an estimate of the time consumed in processing the entire application. Finally, the methodology was able to generate the configuration parameters correctly. The results of the mapping algorithm are promising and provide a proof of concept of the proposed methodology.

4 Conclusions

In this paper, we presented a mapping methodology for coarse-grained pipelined hardware architectures. This automated application mapping methodology provides a reuse capability for this family of high-performance architectures. The methodology is suitable for both offline and run-time mapping as it can provide the set of configuration parameters. Our future work will decrease the execution time of the application by developing an optimization algorithm.

Acknowledgements.

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

References

[1] L. Ma et al., "A graph-based spatial mapping algorithm for a coarse grained reconfigurable architecture template," in *Informatics in Control, Automation and Robotics*, pp. 669–678, Springer, 2012.

[2] S. A. Chin and J. H. Anderson, "An architecture-agnostic integer linear programming approach to cgra mapping," DAC '18, (NY, USA), pp. 128:1–128:6, ACM, 2018.

[3] B. Mei et al., "Dresc: a retargetable compiler for coarse-grained reconfigurable architectures," FPT'02, pp. 166–173, Dec 2002.

[4] J. Bartovský et al., "Morphological co-processing unit for embedded devices," *JRTIP*, pp. 1–12, Jul 2015.