Effective Reconfigurable Design: The FASTER Approach ⋆

D. N. Pnevmatikatos[1], T. Becker[2], A. Brokalakis[8], G. Gaydadjiev[3], W. Luk[2], K. Papadimitriou[1], I. Papaefstathiou[8], D. Pau[7], O. Pell[6], C. Pilato[4], M. D. Santambrogio[4], D. Sciuto[4], and D. Stroobandt[5]

[1] Foundation for Research & Technology - Hellas, GREECE
[2] Imperial College London, UK
[3] Chalmers University of Technology, SWEDEN
[4] Politecnico di Milano, ITALY
[5] Ghent University, BELGIUM
[6] Maxeler Technologies, UK
[7] STMicroelectronics, ITALY
[8] Synelixis, GREECE

Abstract. While fine-grain, reconfigurable devices have been available for years, they are mostly used in a fixed functionality, "asic-replacement" manner. To exploit opportunities for flexible and adaptable run-time exploitation of fine grain reconfigurable resources (as implemented currently in dynamic, partial reconfiguration), better tool support is needed. The FASTER project aims to provide a methodology and a tool-chain that will enable designers to efficiently implement a reconfigurable system on a platform combining software and reconfigurable resources. Starting from a high-level application description and a target platform, our tools analyse the application, evaluate reconfiguration options, and implement the designer choices on underlying vendor tools. In addition, FASTER addresses micro-reconfiguration, verification, and the run-time management of system resources. We use industrial applications to demonstrate the effectiveness of the proposed framework and identify new opportunities for reconfigurable technologies.

1 Introduction

Fine-grain, reconfigurable devices have been available for years in the form of FPGA chips. Many of these devices support the dynamic modification of their programming while they are operating (Dynamic Reconfiguration). However, this ability remains mostly unused as FPGA devices are mostly used in a fixed functionality, "asic-replacement" manner. This is due to the increased complexity in the design and verification of a changing system. In addition to design requirements, the process of creating (partially) reconfigurable designs is less widespread and the corresponding tools are less friendly to the designers.

⋆ The FASTER project is supported by the European Commission Seventh Framework Programme, grant agreement #287804. http://www.fp7-faster.eu/

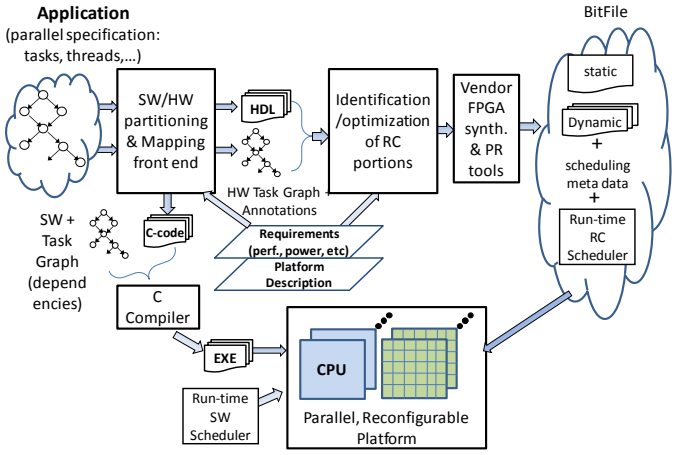

Fig. 1: The FASTER tool-chain

We believe that in order to exploit the opportunities presented by flexible and adaptable exploitation of fine grain reconfigurable resources, better analysis and implementation tool-chains are needed. For example, in a Network Intrusion Detection System, packet contents are compared against suspicious content. This can be efficiently done in FPGAs [1]. New threat identification results in new rules that must be added to the system. Reconfigurable logic allows the incorporation of new functions to the baseline hardware system, combining hardware speed with software-like flexibility.

The FASTER project (http://www.fp7-faster.eu/) aims to provide a methodology and a tool-chain that will enable designers to efficiently implement a reconfigurable system on a platform combining software and reconfigurable resources hiding as much as possible low-level technology details from the user. Figure 1 illustrates the envisioned tool-chain. Starting from a high-level application description and a target platform, our tools analyse the application, evaluate reconfiguration options, alter the application description to include reconfigurability, and implement the designer choices on the underlying vendor tools. In addition, FASTER addresses micro-reconfiguration, a technique to reprogram very small portions of the FPGA when a set of infrequently-changing parameters define the function of a block [2], verification, and the run-time management of system resources. We use industrial applications to demonstrate the effectiveness of the proposed framework and identify new opportunities for reconfigurable technologies.

Technical progress of our work has been documented in technical papers that can be found in http://www.fp7-faster.eu/. Practical achievements include the integration of partial reconfiguration functionality in the Maxeler design flow [3], and its demonstration of partial reconfiguration in a Maxeler platform using a

canny edge detection code, the use of micro-reconfiguration for a NIDS application, and the application of our verification tools on a large scale application (Reverse time migration) [4].

The paper is structured as follows: In Section 2 we discuss related efforts within the context of EU projects and the novel aspects of our work. Section 3 presents the designs methods and how we combine them to form the tool-chain, while Section 4 discusses the way the system is controlled at run-time. Finally Section 5 closes the paper.

2 Related Work and Novelty

Several efforts exist towards similar directions with FASTER project such as the concurrent development of architecture and application for heterogeneous systems. hArtes [5] was an EU-funded project targeting automatic parallelization and generation of heterogeneous systems. It adopted OpenMP pragmas to specify the parallelism automatically extracted from the initial sequential specification but they did not address any aspect related to reconfiguration or dynamic execution. In FASTER we use the same formalism to represent the parallel application, even if the partitioning is provided by the designer since automatic parallelization does not fall into the project's scope. Other EU-funded projects such as REFLECT [6], ACOTES [7] and ANDRES [8] conducted research on the necessary stages of a tool-chain and addressed similar issues with FASTER, but they focused more on system-level or architectural aspects of reconfiguration. Moreover, they did not explicitly emphasize on the design and runtime aspects of partial and dynamic reconfiguration, or, on choosing the best reconfiguration grain-size. Finally, the ERA project [9] adopts dynamic reconfiguration (with low-level OS support) but it targets only a specific platform developed by the consortium.

None of the existing approaches abstracts from the designer complex manipulations needed to control effectively hardware accelerators, in particular when these are designed as dynamically reconfigurable modules. Towards this direction, we aim at providing a general formulation, capable to deal with different multiprocessor systems (targeting the embedded, desktop and high-performance computing domains), supporting different hardware implementations for the tasks and proposing a tool-chain that efficiently partitions the application, while performing considerably more exploration on the possible solutions for the problem. In addition, it takes reconfiguration into account from the early stages of the design process all the way to its runtime use, hiding most of the implementation details from the user.

Other novelties of FASTER are the study of the way micro-reconfiguration is integrated into a tool-chain and interacts with the other tools, and our verification approach which applies equally to static, region-based, or micro-reconfiguration without modification. Finally, we envision a Run-Time System Manager (RTSM) able to support a wide range of platforms, thus we are studying the extent to which it will be developed as a generic library.

3 Methods and Tool-Chain

The starting point of our front-end is a C application, whose initial decomposition is described with OpenMP pragmas, and an XML file containing the target architecture definition (#processing elements, HW/SW tasks characterization, their different implementations and so on). The application task graph is derived and partitioned to determine which processing element (PE) will execute each of the tasks. Our tool-chain performs the following processing steps:

Application profiling and identification of reconfigurable cores: This step analyses the C-code, identifies tasks that could be moved to reconfigurable hardware, and partitions the application accordingly. Based on the initial source code of the application and the description of the target architecture, it decomposes the application into tasks and assigns them to the different components of the architecture. It can also receive information about the achieved performance of the current task assignment, and feedback after the identification of the schedule (e.g. how the partitioning affects the computed schedule) to improve the solution. It also determines (i) the best reconfiguration level (none, region- or micro reconfiguration) for each of the HW cores, and (ii) the properties of the identified tasks, such as the frequency of call functions and parameters changing, the resources required by the implementations, and the execution performance. This processing includes analysis of the call graph, estimation of data transfers, and source code profiling.

High-level Analysis: This step explores various implementation options for applications (or parts of applications) that target reconfigurable hardware and identifies automatically opportunities for run-time reconfiguration. The exploration is performed based on high-level design estimates to avoid time-consuming iterations in the design process, and produces estimates of implementation attributes such as area, computation time, and reconfiguration time; these can be looped back to the *Application profiling and identification of reconfigurable cores* step, to perform iterative design optimizations for arithmetic operations presentation, computational precision, and parallelism in the implementation. The High-level Analysis also provides an automatic way to suggest opportunities for reconfiguration, such as partitioning of the application into several reconfigurable components.

Optimizations for region- and micro-reconfiguration: This step receives the descriptions of the tasks, i.e. the corresponding source code, that could benefit from the reconfiguration and it produces new and optimized implementations for them to be considered for the task mapping. This analysis will be performed also through dynamic profiling of the application tasks to determine the parameters for the micro-reconfiguration and through the identification of isomorphic subgraphs for supporting the data-path merging and thus reducing the number of reconfigurations.

Compile-time scheduling and mapping onto reconfigurable regions: It receives information about the application and the architecture from the two previous processing steps, focusing on the tasks assigned to the reconfigurable hardware,

and it determines their scheduling along with the mapping of the cores onto the reconfigurable regions. In particular, it determines the number and the characteristics (e.g. size) of these regions, the number and the size of each input/output point, and also takes into account the interconnection infrastructure of the system (e.g. bus size). Also, it schedules the resulting implementation and annotates the characterization part with such information to further refine the specification. It annotates the tasks with information about the feasibility of the implementation where the solution is specified (i.e. if the reconfigurable region can satisfy the resource requirements) and it provides feedback to the partitioning methodology to further refine the solution.

Verification: To verify that a simple, unoptimized design (the source) implements the same behaviour as an optimized, possibly reconfiguring design (the target), we combine symbolic simulation with equivalence checking. The source and target designs are first compiled for a symbolic simulator, which then stimulates the design with symbolic inputs, rather than the numerical or Boolean inputs used in traditional approaches. Equivalence checking is used to check symbolic outputs from source and target designs that may differ but still be equivalent (for example $b + a$ instead of $a + b$). If symbolic outputs from source and target designs are equivalent for all inputs, the designs are proved equivalent, otherwise, the first input with different outputs can be used to debug the target design.

4 Run-Time System

The Run-Time System Manager (RTSM) is responsible for managing resources, scheduling SW and HW tasks, and enforcing adaptation of the system according to functional and non-functional parameters (e.g. temperature) for applications developed with the FASTER tool-chain. Its basic components and functionality were presented abstractly in [10]. The concept of its development relies on the work initially published in [11]. Here we describe briefly its first implementation, which takes into account all known restrictions imposed by the current PR technology.

Figure 2 illustrates how the RTSM is generated, its basic components and actions. RTSM basic functionality is specified by the baseline scheduler contained in the XML file. The XML contains the available runtime alternatives to reconfigure the regions, the representation of the task graph, the number of iterations for each task, and additional information about each task which will be used for the scheduling, e.g. power consumption, reconfiguration time, execution time. The necessary information retrieved from the XML file is used for feeding the RTSM structures.

5 Conclusions

The FASTER project enhances various aspects in designing modern computing systems. The main challenge is the inclusion of reconfiguration as an explicit

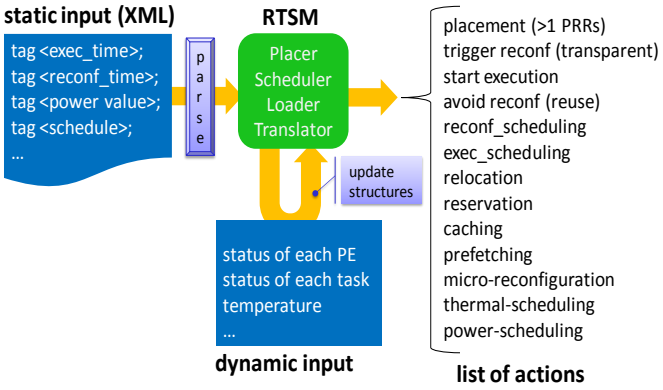

Fig. 2: RTSM inputs, characteristics and operations

design concept. To do this we are developing new design methods and a toolchain for efficient and transparent use of reconfiguration.

References

1. I. Sourdis, D. Pnevmatikatos, and S. Vassiliadis, "Scalable Multi-Gigabit Pattern Matching for Packet Inspection," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 2, pp. 156–166, February 2008.
2. K. Bruneel and D. Stroobandt, "Automatic Generation of Run-Time Parameterizable Configurations," in *IEEE International Conference on Field Programmable Logic and Applications (FPL)*, August 2008, pp. 361–366.
3. R. Cattaneo, C. Pilato, M. Mastinu, O. Kadlcek, O. Pell, and M. D. Santambrogio, "Runtime Adaptation on Dataflow HPC Platforms," in *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, June 2013.
4. T. Todman and W. Luk, "Verification of Streaming Designs by Combining Symbolic Simulation and Equivalence Checking," in *IEEE International Conference on Field Programmable Logic and Applications (FPL)*, August 2012.
5. http://www.hartes.org/, [Online; accessed 2012].
6. http://www.reflect-project.eu/, [Online; accessed 2012].
7. http://www.hitech-projects.com/euprojects/ACOTES/, [Online; accessed 2014].
8. http://andres.offis.de/, [Online; accessed 2014].
9. http://www.era-project.eu/, [Online; accessed 2014].
10. D. Pnevmatikatos, T. Becker, A. Brokalakis, K. Bruneel, G. Gaydadjiev, W. Luk, K. Papadimitriou, I. Papaefstathiou, O. Pell, C. Pilato, M. Robart, M. D. Santambrogio, D. Sciuto, D. Stroobandt, and T. Todman, "FASTER: Facilitating Analysis and Synthesis Technologies for Effective Reconfiguration," in *Euromicro Conference on Digital System Design (DSD)*, September 2012.
11. G. Durelli, C. Pilato, A. Cazzaniga, D. Sciuto, and M. D. Santambrogio, "Automatic Run-Time Manager Generation for Reconfigurable MPSoC Architectures," in *IEEE International Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, July 2012.