2011 21st International Conference on Field Programmable Logic and Applications

A Hybrid Mapping-Scheduling Technique for Dynamically Reconfigurable Hardware

Juan Antonio Clemente[*]

Vincenzo Rana[#^]
Donatella Sciuto[#]

Ivan Beretta[^]
David Atienza[^]

[*] DACYA
Universidad Complutense de Madrid
(UCM)
Madrid, 28040, Spain
ja.clemente@fdi.ucm.es

[#] Dipartimento di Elettronica
e Informazione (DEI)
Politecnico di Milano
Milano, 20133, Italy
{rana,sciuto}@elet.polimi.it

[^] Embedded Systems Lab (ESL)
Ecole Polytechnique Federale
de Lausanne (EPFL)
Lausanne, 1015, Switzerland
{ivan.beretta,david.atienza}@epfl.ch

***Abstract*— Reconfigurable computing is a promising technology that offers an interesting trade-off between flexibility and performance, which many recent multi-core embedded system applications demand. In order to achieve these objectives, it is necessary to optimize the deployment of the hardware cores on the FPGA platform, trying to reduce the reconfiguration overhead while meeting the desired performance. In this paper, we propose a hybrid mapping and scheduling technique for multi-core applications on reconfigurable devices, which exploits the information about the relationships among the application cores to minimize the overhead due to reconfiguration.**

I. INTRODUCTION

In the last few years, dynamically reconfigurable devices, such as FPGAs, have become popular since they combine both the performance requirements and the flexibility that many applications in embedded systems demand. However, a well-known drawback of dynamic reconfiguration is the timing overhead related to reconfiguration processes, which can also reach hundreds of milliseconds. This could negatively affect the performance of the applications deployed on the system, especially when reconfigurations are very frequent. Hence, good mapping and scheduling strategies are needed in order to assign the cores to the reconfigurable resources and to schedule their reconfigurations, minimizing the total reconfiguration overhead while guaranteeing the required performance.

In this paper we propose a hybrid mapping-scheduling technique able to exploit the spatial and temporal information of the cores of the incoming applications, taking into account both the performance of the whole system and the timing overhead related to dynamic reconfiguration processes.

We applied both the proposed representation model and the proposed hybrid mapping-scheduling technique to a real-world case study, the MP3 decoder [1], and we deployed it on a hardware platform based on a Xilinx Virtex-5 XC5VLX110T FPGA. We also applied the proposed approach to a set of 50 synthetic benchmarks characterized by a different number of cores in order to validate that the proposed approach scales very well with the complexity of the target applications. Experimental results show that the proposed approach is able to save up to 30% of resources and to halve the reconfiguration overhead with respect to other state-of-the-art techniques [2] on the synthetic benchmarks, while achieving even better results on the analyzed real-world case study.

This research has been partly supported by the HiPEAC network of excellence, by the Swiss National Science Foundation (SNF), grant number 200021-127282, and by the Spanish Department of Education and Science under grants TIN2009-09806 and AYA2009-1330-C03-02.

II. RELATED WORK

This section analyzes the different approaches that can be found in literature to solve the problem introduced in Section I. Optimal algorithms based on Integer Linear Programming (ILP) [3], as well as heuristic approaches [4], have been proposed to schedule Data Flow Graphs (DFGs) onto reconfigurable systems. Hybrid mapping-scheduling techniques for DFGs can also be found in the literature. For instance, [5] presents a Mixed Integer Linear Program (MILP) based model to tackle mapping and scheduling at the same time. However, either these approaches specifically do not take into account the impact of the reconfiguration overhead, or they cannot explicitly handle situations in which several tasks must coexist at the same time (pipelined behavior), which frequently occurs in reconfigurable systems.

Other approaches aim at finding a suitable mapping of applications specified as Data Flow Diagrams (DFDs). An example can be found in [6], which however does not explicitly take into account the possibility of dynamically reconfiguring the cores. Reconfiguration is considered in [2], where the authors aim at minimizing the switching time between two subsequent mappings. However, the solution could be generally considered as sub-optimal, because the available scheduling information is not exploited.

The approach proposed in this article is able to explicitly handle the reconfiguration overhead, and to capture all the relevant scheduling information, such as a pipelined behavior between the cores. The work has been compared with the algorithm presented in [2], since it is the only one that takes the impact of dynamic reconfigurations into account.

978-0-7695-4529-5/11 $26.00 © 2011 IEEE
DOI 10.1109/FPL.2011.40

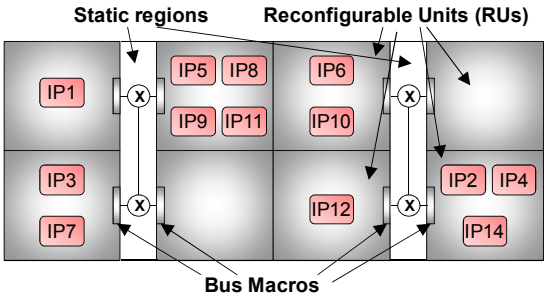

Figure 1. The target architecture

III. TARGET ARCHITECTURE

This section defines the partially reconfigurable architecture targeted in this work and shown in Fig. 1, which consists of a set of static regions, as well as several Reconfigurable Units (RUs). Each static region contains an inter-region communication infrastructure based on a Network-on-Chip (NoC), optimized for FPGA implementations. The reconfigurable regions are filled up with cores (along with their interfaces towards the communication infrastructure). Depending on their size, multiple cores can be assigned to the same RU, thus forming an island of cores [7]. The intra-region communications among these cores are resolved internally by means of a sub-NoC that is reconfigured along with the cores. Thus, the intra- and inter-region NoCs can be considered a unique 2-level NoC, which is contention-aware and provides enough bandwidth with respect to the communication needs of all the target applications. For this reason, the proposed mapper-scheduler can safely ignore placement. It is also worth noting that the inter-region NoC is essentially static, thus it is fully operational even during the reconfiguration of a RU, as well as the communication channels between the reconfigurable and the static regions. In Virtex devices, this is achieved by means of bus macros, which are placed along the edges of the reconfigurable regions.

IV. HYBRID MAPPER-SCHEDULER

The proposed Hybrid Mapper-Scheduler (HMS) receives as input an application, which is specified in such a way to include its deadline, its cores, the communication links among them, and their lifetimes. A lifetime is defined as the time interval when a core needs to be configured on the device in order to correctly contribute to the execution of the whole application.

In particular, the proposed HMS aims at finding a temporal assignment of the cores to the different islands of the target architecture, trying to reduce both the reconfiguration overhead and the resources usage, while meeting the specified deadline.

A. Pre-processing

During the first stage, the algorithm divides the input application into a set of snapshots (Step 1, Fig. 2), i.e., the time intervals when the set of cores that must be configured at the same time does not change. The snapshots are generated by sorting these time instants decreasingly, and by selecting two consecutive times from this sorted list.

Since the cores in each snapshot do not change, it is possible to translate each snapshot into a traditional DFD, where the weight of each node represents the cores size, and the weight of each link is the amount of traffic between two cores. In order to guarantee that the application can be deployed on the FPGA, it is sufficient to guarantee that each snapshot generated from the original application fits into the target architecture.

Once the snapshots have been generated, the proposed HMS generates a first solution (Step 2). The solution is computed by mapping all the snapshots separately, i.e., the cores of each snapshot are assigned to a RU to form a set of islands. The algorithm we employed to generate the islands is based on graph partitioning theory, and is implemented by an external tool named CHACO [8]. Each DFD associated with a snapshot is given as an input to CHACO and it is partitioned into a given number of parts: since each part contains one or more cores, it corresponds to an island in the hardware architecture. The partition found by CHACO is said to be feasible if the total size of each island is lower than the size of a RU. However, CHACO generates a partition of the input DFD aiming at balancing the size of each part, thus it has the tendency to uniformly spread the cores over the specified number of islands, rather than to minimize the number of islands itself. This behavior may negatively affect the quality of the solution. In order to overcome this limitation, the proposed HMS executes CHACO multiple times, iteratively increasing the number of islands until a feasible partition is found.

B. Mapping-Scheduling of the Application

After the first solution has been generated, the application is essentially decomposed into a series of consecutive DFDs that must be deployed sequentially. The rest of the algorithm tries to recombine them in order to minimize the average transition time between two consecutive snapshots.

The algorithm evaluates the quality of the first solution (Step 3) by estimating its overall execution time, which includes the reconfiguration overhead that is necessary to load the cores required by each snapshot, and stores it (Step 4). The execution time is used to evaluate every intermediate solution during the execution of the HMS algorithm (Step 5), which iteratively refines the solution (Steps 6-13) until the deadline associated to the application is met (Step 14). The iterative refinement aims at reducing the execution time by smoothing the transitions among the snapshots. The idea is to identify two consecutive partitioned snapshots which are similar enough to allow the algorithm to merge them, and form a set of islands that can be used by both the snapshots, thus eliminating the transition between them (we call it *mergeable* transition), and also avoiding any reconfiguration when switching from one snapshot to the other. Once the transition between two partitioned snapshots P_i and P_{i+1} has been merged, any future merging operation involving P_{i+1} will also involve P_i as well, and viceversa.

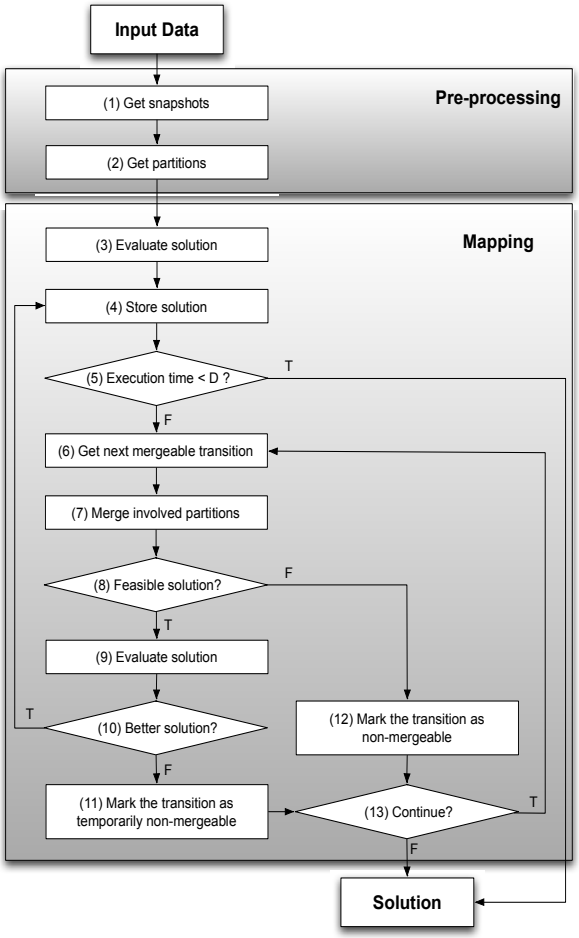

Figure 2. Flowchart of the proposed HMS algorithm

The merging operation aims at making two consecutive partitioned snapshots P_1 and P_2 compatible. The compatibility condition ensures that the cores that are in common between the two snapshots are mapped in islands that can be used by both the snapshots, thus avoiding any unnecessary reconfiguration. In particular, P_1 and P_2 are compatible if:

$$(1) \quad \forall i, j \in Cores_{P1} \cup Cores_{P2} \left[I(i) = I(j)\right]_{P1} \Leftrightarrow \left[I(i) = I(j)\right]_{P2}$$

This expression means that, for all the cores in P_1 and P_2, if cores i and j belong to the same island in the partition P_1, they have to belong to the same island in P_2 and viceversa ($I(i)$ indicates the island to which the core i belongs). If P_1 and P_2 are compatible, the islands of P_1 can be used by P_2 without introducing any reconfiguration overhead, while a reconfiguration is needed for the islands containing the cores required by P_2 but not by P_1.

We show how two snapshots can be made compatible using the example in Fig. 3, which refers to the MPEG-4 Layer-2 Simple Profile (SP) encoder [1]. This application contains four cores, namely the Variable Length Decoder (VLD), the Inverse Discrete Cosine Transform (IDCT), the Motion Compensation (MC) and the Rate Control (RC). The processes IDCT and MC

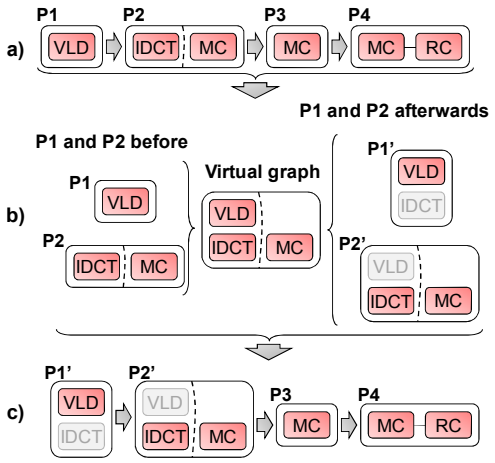

Figure 3. Merging process of P1 and P2 in MPEG-4. The dashed lines divide different islands of the same partition

run after VLD, whereas the MC and RC are performed in a pipelined way until the complete stream of images is processed, and therefore they must be configured at the same time on the device.

Fig. 3a shows the initial partitioning of the MPEG-4 application: in this example, the goal is to merge the partitioned snapshots P_1 and P_2. Firstly, the proposed algorithm creates a virtual graph (VG) according to the following condition:

$$(2) \quad \forall i \in Cores, \left(i \in P_1 \vee i \in P_2 \Leftrightarrow i \in VG\right)$$

Thus, if a core exists in either P_1 or P_2, it is included in VG. Then, the algorithm uses CHACO to partition VG, and it starts from its output to create the new islands using the following rule:

$$(3) \quad \forall i \in \{P_1, P_2\}, \forall j \in Islands_{VG} \left[\exists z \in Cores(j) : used(z, i) \Rightarrow j \in P_i'\right]$$

This expression means that, for each snapshot i and for each island j of the partitioned VG, if j contains at least one core z that is used in snapshot i, then the island should be used by the merged snapshot P_i'. Using this technique, the new snapshots satisfy the compatibility condition equation (1) by construction: in fact, all the cores that are in common are assigned to one island that will be used by both the snapshots, and therefore does not need to be reconfigured. In Fig. 3b, the partitioning of the VG generates two islands: one contains cores VLD and IDCT, while the other contains MC. The first island is selected to belong to P_1' because VLD belongs to the first snapshot, and this is done even though IDCT does not. As a consequence (Fig. 3c), the IDCT will be loaded with the first snapshot, but it will not be used (in the picture, IDCT appears shaded). However, when the second snapshot P_2' is executed, the IDCT is already configured on the device and no reconfiguration is needed (on the other hand, the second snapshot keeps VLD configured but does not use it), while only the island containing MC needs to be loaded. Finally, it is important to remark that a merging operation may create large islands that cannot fit into the RUs: for this reason, the

algorithm always has to check the feasibility of the solution (Step 8).

In order to perform the aforementioned merging procedure, the algorithm identifies the transition between two consecutive snapshots that generates the greatest reconfiguration overhead (Step 6) and tries to eliminate it by merging the two snapshots (Step 7). If the merging operation does not generate a feasible solution, the transition is marked as non-mergeable (Step 12), and it will not be considered again. Otherwise, the solution is evaluated (Step 9) and, if its execution time is lower than the best one found so far (Step 10), it becomes the new best solution (Step 4). The procedure stops as soon as the deadline associated with the application is met (Step 5). If the merging operation does not lead to an improvement in terms of execution time, the transition is temporarily marked as non-mergeable (Step 11), but it will be considered again whenever one of the two snapshots is involved in another merging operation.

Every time a merging operation has been completed (either successfully or not), the algorithm checks if the deadline constraint is met and, if it is not, looks for another transition that can be potentially optimized (Step 13).

The proposed algorithm is structured in such a way that the more snapshots are merged, the more global the optimization is. In fact, each merging process reduces the reconfiguration overhead, although more hardware resources are used to load cores that may not be needed by all the snapshots. As a consequence, the more snapshots are merged, the more area is used, and for this reason the proposed HMS stops as soon as the temporal constraints are met.

V. Performance Evaluation

This section presents several different experiments that have been carried out both to validate the proposed approach and to evaluate the quality of the mapping-scheduling solutions. In particular, the proposed HMS approach has been applied to a complex real-world case study: an MP3 decoder consisting of 15 cores [1].

The proposed HMS approach has also been applied to a set of 50 synthetic benchmarks in order to evaluate how it scales when the complexity of the target application increases. These synthetic benchmarks consist of 30 cores (with sizes ranging from 100 to 500 slices), each one associated with different lifetimes (from 1 to 5 disjoint lifetimes per core) that span from 0 to 100 ms, in such a way that each benchmark is composed of 10 snapshots. These benchmarks have been deployed on a system based on a Virtex 5 XC5VLX110T FPGA, where tasks can run up to 178 MHz. According to our measurements, each RU takes approximately 4 ms to be reconfigured (when the system consists of 9 RUs, with 622 slices per RU).

In order to evaluate the quality of the solutions obtained with the proposed HMS approach, we compared its results, in terms of performance, with those obtained by the most similar approach that can be found in literature, i.e., the one presented in [2]. Fig. 4 shows the reconfiguration overhead obtained by using different values (ranging from 0 to 40 ms) of the *extended deadline* parameter, which represents the maximum

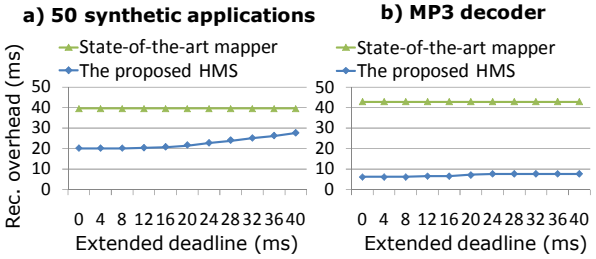

Figure 4. Reconfiguration overhead of the proposed HMS approach and of the approach proposed in [2]

acceptable delay with respect to the optimal execution time of the application. These experiments have been carried out with a varying number of RUs (ranging from 3 to 9) and then averaging the obtained results.

As shown by Fig. 4a, the proposed HMS approach outperforms the state-of-the-art one, achieving a reduction of the reconfiguration overhead up to 49% when the extended deadline is set to 0 ms. A similar trend can also be observed in Fig. 4b, which shows that the proposed approach is able to reduce the reconfiguration overhead up to 86% when the extended deadline is set to 0 ms.

VI. Concluding Remarks

In this paper we have proposed a hybrid mapping-scheduling technique able to reduce both the impact of the dynamic reconfiguration overhead and the reconfigurable resources usage. Experimental results have shown that the proposed HMS algorithm outperforms the most similar approach that can be found in the literature [2], saving up to 30% of the reconfigurable resources and almost halving the reconfiguration overhead both over a wide set of benchmarks and on real-world case studies.

References

[1] B.D. Theelen et al., "Scenario-Aware Dataflow". Technical Report ESR-2008-08, Eindhoven University of Technology, 2008.

[2] V. Rana et al., "Minimization of the reconfiguration latency for the mapping of applications on FPGA-based systems". Proc. of CODES+ISSS, Oct. 2009, pp. 325 – 334.

[3] S. Ghiasi, A. Nahapetian and M. Sarrafzadeh, "An optimal algorithm for minimizing run-time reconfiguration delay". ACM Trans. in Embedded Computing Systems, vol. 3, no. 2, pp. 237 – 256, May 2004.

[4] J. A. Clemente, J. Resano, C. González and D. Mozos, "A Hardware Implementation of a Run-Time Scheduler for Reconfigurable Systems". IEEE Trans. on Very Large Scale Integration (VLSI) Systems, vol. 19, no. 2, pp. 1263 – 1276, July 2011.

[5] A. Bender, "MILP based task mapping for heterogeneous multiprocessor systems". In Proc. of EURO-DAC, Sept. 1996, pp. 190 – 197.

[6] S. Murali, M. Coenen, A. Radulescu, K. Goossens and G. De Micheli, "A methodology for mapping multiple use-cases onto networks on chips". In Proc. of DATE, Sept. 2006, pp. 118 – 123.

[7] I. Beretta, V. Rana, D. Atienza and D. Sciuto, "Island-Based Adaptable Embedded System Design", IEEE Embedded Systems Letters, vol. 3, no. 2, pp. 53 – 57, June 2011.

[8] B. Hendrickson and R. Leland, "The Chaco User's Guide: Version 2.0". Sandia Tech Report SAND94-2692, 1994.