A Multiobjective Reconfiguration-Aware Scheduler for FPGA-Based Heterogeneous Architectures

Enrico A. Deiana, Marco Rabozzi, Riccardo Cattaneo, Marco D. Santambrogio
Politecnico di Milano, Milan, Italy
enrico.deiana@mail.polimi.it, {marco.rabozzi, riccardo.cattaneo, marco.santambrogio}@polimi.it

***Abstract*—Designing applications for heterogeneous systems, like Multiprocessor System-on-Chips (MPSoCs) based on Field Programmable Gate Arrays (FPGAs) is a complex task. In order to exploit all the capabilities of these systems, such as Partial Dynamic Reconfiguration (PDR) and hardware acceleration, the designer still has to develop large parts of the system unassisted, establishing the design choices (i.e., whether to assign a task of the application on a hardware region of the FPGA or a general purpose processor of the SoC) mostly on his/her experience.**

In this paper we present a Mixed-Integer Linear Programming (MILP) formulation for mapping and scheduling of applications on heterogeneous and reconfigurable devices taking into account PDR, module reuse and configuration prefetching. Starting from a target architecture and a description of the application in terms of tasks and data dependencies, the proposed formulation allows the designer to optimize a linear combination of different metrics such as execution time, peak power and energy consumption.

I. Introduction

The evolution of heterogeneous architectures such as FPGA-based MPSoCs is placing new challenges for system designers, due to the fact that many features and techniques have to be taken into account in order to achieve an acceptable application design, related, in our case, to the phases of mapping and scheduling of the application tasks. Among these features and techniques there is *PDR* [1], which is the possibility to change a portion of the FPGA configuration at runtime. If we look at the FPGA as a matrix, such change can affect the whole height of the FPGA with respect to the region that has to be reconfigured (1D reconfiguration) or only part of it, leading to a smaller and more precise reconfiguration (2D reconfiguration). The reconfiguration time is directly proportional to the bitstream size of the area that has to be reconfigured, the smaller the area the quicker is the reconfiguration. Hence, *module reuse* [2] (i.e., sharing resources among equal tasks that have already been placed on the FPGA) is desirable in order to reduce the number of reconfigurations. Another important feature is *reconfiguration prefetching*, which allows the configuration of a task to be performed ahead of time, prior to actually requiring the corresponding task execution, hence the reconfiguration time can be hidden more effectively by the execution of other tasks.

Since the better these techniques and features are exploited the better is the resulting solution, and taking into account that the space of the solutions is considerable, it is necessary the use of automatic tools in order to help the system designer in building the best possible application schedule with respect to his/her needs. Such requirements refer not only to the overall execution time (like most of the other schedulers in the literature do [2]–[8]), but also to power and energy consumption of the design. This means addressing a specific Resource Constrained Scheduling Problem (RCSP) [9], that takes into consideration the reconfiguration aspects with a focus on power and energy metrics that are becoming crucial in designing applications targeting heterogeneous architectures. Following this view we propose an iterative off-line scheduler, based on a MILP model, which provides the following contributions:

- the possibility to tune different performance metrics among execution time, peak power and energy consumption, or a linear combination of them;
- the consideration of the features and techniques offered by current FPGA-based devices (PDR, module reuse, reconfiguration prefetching);
- the opportunity to easily trade-off the quality of the desired scheduling solution with respect to the execution time of the algorithm.

The remainder of the paper is organized as follows: Section II shows the related works in the literature, Section III presents a description of the problem and the proposed approach, Section IV discusses the proposed MILP model, Section V shows how the model is exploited within the iterative scheduling algorithm, Section VI evaluates our approach on different problem instances and, finally, Section VII draws the conclusions.

II. Related work

In literature there are many approaches addressing variants of the RCSP problem that rely on heuristic algorithms ([2], [5]–[7], [10], [11]) and exact algorithms ([2]–[4], [8], [12]). What follows is a description of these works in which we highlight their contributions and limitations.

Among the variety of heuristic approaches we report the work in [4] that uses a KLFM-based heuristic for simultaneous mapping and scheduling of tasks on heterogeneous architectures. Still, it does not take into account module reuse and it only considers architectures with 1D reconfiguration, which is a limitation with respect to the potentialities of current available FPGAs that feature 2D reconfiguration. The same authors present in [6] a greedy scheduler called PARLGRAN that tries to add as many copies of data parallel tasks as it can, in order to maximize the number of concurrent operations. However, as in the previous work, the reuse of

978-1-4673-9406-2/15/$31.00 ©2015 European Union

tasks configurations is not exploited. On the other hand, in [7] it is proposed a heuristic for the mapping of tasks on hardware and software components, which is based on a genetic algorithm and a list-based scheduler. This approach does not take into consideration reconfiguration prefetching. An original idea, that focuses on reducing communication overhead for scheduling applications, is proposed in [5]. It applies clustering techniques to group tasks that exchange a high amount of data in order to reduce the transfer overhead, yet it is not addressing PDR. Another approach is suggested in [2], where, by means of a reconfiguration-aware scheduler called Napoleon, it targets FPGA-based architectures with more than one 2D reconfigurator and takes into account both module reuse and reconfiguration prefetching, but it only optimizes the schedule duration. A last noteworthy heuristic is PaRA-Sched [10], a scheduler that takes advantage of Ant Colony Optimization (ACO). The algorithm considers all the previous cited features and techniques during the design space exploration. Nevertheless, as it will be shown in this paper, the quality of its solutions can still be improved.

Among the exact approaches, the authors in [3] propose an algorithm that models the communication among hardware components, adding the time spent for the exchange of information to the task execution time. The applicability of this solution is limited by the following assumptions: any number of reconfigurations can take place at the same time and reconfiguration prefetching is not allowed. In [4] an Integer Linear Programming (ILP) formulation is introduced. It considers many aspects of heterogeneous architectures, such as reconfiguration prefetching, simultaneous mapping and scheduling of tasks on both hardware and software components and also PDR. Nevertheless, the model makes use of 1D reconfiguration only. Another ILP model is described in [8]. Even though it addresses PDR, each task of the considered applications is assumed to occupy the same area on the chip (simplifying the mapping phase) and to have negligible delays compared to the reconfiguration ones. Overcoming the previous limitations, the authors in [2] present an ILP formulation that is able to model multiple 2D reconfigurators. However, the approach is limited to time domains discretized into a small number of units due to the considerable amount of binary variables required.

It is also worth to notice that all the previous proposed solutions currently use as quality metrics the schedule execution time only, while other performance metrics such as peak power or energy consumption are not considered. On the other hand, [12] and [11] focus on power minimization. [12] uses an ILP model with time constraints on the schedule execution time and targets heterogeneous architectures composed by a General Purpose CPU (GPCPU) and co-processors, while [11] exploits cluster techniques for mapping operations to microprocessors or Application Specific Integrated Circuits (ASICs). However, both approaches to not reconfigurable logic (i.e., FPGAs) and PDR. Most of the previous works do not take into account all the three main features of current FPGAs discussed in the previous section: PDR, module reuse and reconfiguration prefetching. Those which do ([2], [10]) have limitations related to the quality of the solution or the time needed to obtain the result.

TABLE I
SCHEDULER INPUT DATA

Target architecture description	
C^s	set of available software components (i.e., processors)
C^h	set of hardware components (i.e., reconfigurable regions on the FPGA)
C	set of all components ($C^s \cup C^h$)
R	set of FPGA resources (i.e., CLB, DSP, BRAM, ...)
bit_r	average bitstream size for a resource of type r
T_{rec}	reconfiguration time for each unit of bitstream
P_{rec}	average power consumption for reconfiguration
Taskgraph representation	
T	set of tasks to schedule
P	set of tasks precedences: $(t1, t2) \in P$ if $t2$ depends on $t1$
Tasks implementations	
I^s	set of software implementations for all tasks T
I^h	set of hardware implementations for all tasks T
I	set of all implementations ($I^s \cup I^h$)
TIC	set of valid mappings: $(t, i, c) \in TIC$ if task $t \in T$ can be mapped on component $c \in C$ with implementation $i \in I$
$time_i$	execution time of implementation $i \in I$
$power_i$	power consumption of implementation $i \in I$
$energy_i$	energy consumption of implementation $i \in I$
$res_{i,r}$	resources of type $r \in R$ required by implementation $i \in I^h$

III. PROBLEM DESCRIPTION AND PROPOSED APPROACH

The input of the scheduling on heterogeneous architectures problem consists of: (1) an architectural template (e.g., ZedBoard with Zynq™-7000 AP SoC) containing information on the General Purpose Processors (e.g., dual core ARM Cortex-A9) and the programmable logic (e.g., Xilinx XC7Z020), (2) a description of the application in terms of a taskgraph, that is a Directed Acyclic Graph (DAG) where nodes represent tasks of the application and the directed edges represent the data dependencies among tasks. Each task can have several hardware and software implementations with different execution time, energy and power consumption. The hardware implementations are also characterized by the resources they need on the programmable logic, such as the number of BRAMs, CLBs, DSPs. A complete formalization of the discussed input data is presented in Table I.

The goal of the scheduler is to assign each task to an implementation, map it on a hardware component (i.e., a reconfigurable region on the programmable logic) or a software one (i.e., one of the available processors) and then schedule it in a time slot according to a user defined objective function. Moreover, the scheduler must take into account the reconfigurations that may be required among subsequent tasks mapped on the same reconfigurable region.

Within this paper we propose an iterative approach based on a MILP formulation of the problem that simultaneously maps and schedules the tasks of the application. Our approach considers the reconfigurations as dynamic tasks with their own dependencies that are added to the taskgraph when required. In this fashion, module reuse and reconfiguration prefetching are directly taken into account as in [10].

TABLE II
MILP VARIABLES AND ADDITIONAL SETS

Additional sets	
RT	reconfiguration tasks
AT	all tasks ($RT \cup T$)
TI	couples (t, i) such that task $t \in T$ can use implementation i
TC	couples (t, c) such that task $t \in T$ can be mapped on component c
CP	couples of tasks $(t1, t2) : t1, t2 \in T$ such that it is possible to schedule $t2$ right after $t1$ on the same hardware component
OT	couples of tasks $(t1, t2) : t1, t2 \in AT$ such that there exists a schedule in which $t1$ and $t2$ overlap in time
CT	couples of tasks $(t1, t2) : t1, t2 \in T$ such that both tasks have at least a common hardware implementation
Real non negative variables	
b_t	the begin time of task t, $\forall t \in AT$
e_t	end time of task t, $\forall t \in AT$
$oc_{c,r}$	the amount of resources of type r needed by hardware component c, $\forall c \in C^h, r \in R$
$bitc_c$	bitstream size for hardware component c, $\forall c \in C^h$
$mibo_{t1,t2,i}$	forced to 1 if $bo_{t1,t2} = 1$ and $mi_{t2,i} = 1$ (variable required for linearization), $\forall (t1, t2) \in OT, (t2, i) \in TI$
Binary variables	
$mic_{t,i,c}$	set to 1 if task t is mapped to component c with implementation i, $\forall (t, i, c) \in TIC$
$mi_{t,i}$	set to 1 if task t is assigned to implementation i, $\forall (t, i) \in TI$
$mc_{t,c}$	set to 1 if task t is mapped to component c, $\forall (t, c) \in TC$
$cp_{t1,t2}$	set to 1 if task $t2$ is executed right after task $t1$ on the same hardware component, $\forall (t1, t2) \in CP$
$cft_{t,c}$	set to 1 if task t is the first task executed on hardware component c, $\forall (t, c) \in TC : c \in C^h$
$rtt_{rt,t}$	set to 1 if task t requires reconfiguration rt prior to its execution, $\forall t \in T, rt \in RT$
$rtc_{rt,c}$	set to 1 if reconfiguration task rt is performed on hardware component c, $\forall rt \in RT, c \in C^h$
$ba_{t1,t2}$	set to 1 if task $t1$ begins after the beginning of task $t2$, or at the same time of $t2$, $\forall (t1, t2) \in OT$
$bb_{t1,t2}$	set to 1 if task $t1$ begins before the end of task $t2$ or at the end of $t2$, $\forall (t1, t2) \in OT$
$bo_{t1,t2}$	set to 1 if the beginning of task $t1$ overlaps in time with task $t2$ (i.e., task $t1$ begins when $t2$ is in execution), $\forall (t1, t2) \in OT$

It is worth noting that if the final solution includes tasks mapped on the reconfigurable hardware, a subsequent floorplanning of the reconfigurable regions is required [13]. In order to ease floorplanning, within the scheduling algorithm we round up the resource requirements of the hardware implementations taking into account the granularity of the minimal reconfigurable unit on the selected FPGA [14]. Furthermore, if the final solution cannot be successfully floorplanned, the scheduler is re-executed virtually reducing the number of available resources until a feasible floorplan is identified. If each task admits at least a software implementation, the latter approach is guaranteed to find a feasible scheduling.

IV. MILP FORMULATION

Within this section we show how the previous problem description has been translated into a suitable MILP model. The set and parameters previously defined in Table I are augmented with some extra sets from Table II that are required for the definition of the formulation. Furthermore, Table II also report the list of variables used within the MILP model grouped by variable type.

One of the main addition with respect to the original problem description is the definition of set RT that is used to take into account potential reconfiguration tasks that might be needed in the schedule. Since we do not not know beforehand which is the number of reconfiguration tasks that will be performed, we consider the worst case scenario in which each task, except for the first one, requires a reconfiguration before its execution. Overall, within set RT we consider $|T| - 1$ elements, while a special binary variable ($rtt_{rt,t}$) is used to determine if the element represents a required reconfiguration or not. Most of the variables listed in Table II are support variables that are needed to ensure the semantics of the model, but are redundant with respect to the definition of a schedule. Indeed, a solution can be fully determined by specifying the begin time b_t and end time e_t of each task (including reconfiguration tasks), the selected implementation and component $mic_{t,i,c}$ for each task, and, the binding between reconfiguration tasks and application tasks $rtt_{rt,t}$.

Due to limited space and for the sake of clarity, we report here only the main constraints of the model[1]. By means of the following constraints, the MILP model:

- ensures the dependencies among the tasks

$$\forall (t1, t2) \in P : b_{t2} \geq e_{t1} \tag{1}$$

- avoids overlap among tasks mapped on the same component (we exploit the fact that if $bo_{t1,t2} = bo_{t2,t1} = 0$ there is no overlapping among tasks $t1, t2 \in OT$)

$$\begin{gathered} \forall (t1, t2) \in OT, \forall c \in C \mid (t1, c), (t2, c) \in TC : \\ bo_{t1,t2} + mc_{t1,c} + mc_{t2,c} \leq 2 \end{gathered} \tag{2}$$

- ensures non overlapping also with respect to cp (i.e., given a task t, there is at most one previous task and one subsequent task on the same hardware component)

$$\begin{gathered} \forall t \in T : \\ \sum_{(t,t2) \in CP} cp_{t,t2} \leq 1 \qquad \sum_{(t2,t) \in CP} cp_{t2,t} \leq 1 \end{gathered} \tag{3}$$

- avoids overlapping between the potential reconfiguration tasks by enforcing a sequential order (to state this inequality we assume the reconfiguration tasks assigned to unique natural numbers in the interval $[1, |T| - 1]$)

$$\forall rt \in RT \mid rt > 1 : b_{rt} \geq e_{rt-1} \tag{4}$$

- ensures that the hardware components do not exceed the resources provided by the FPGA

$$\forall r \in R : \sum_{c \in C^h} oc_{c,r} \leq maxRes_r \tag{5}$$

- ensures that between two subsequent tasks $t1, t2 \in T$ mapped on the same hardware component with different implementations a reconfiguration must be performed to configure task $t2$

[1]For a complete description we refer the reader to the extended formulation available at: http://home.deib.polimi.it/santambr/prj/scheduler/scheduler.htm

$$\forall (t1,t2) \in CP, \forall i1 \in I^h \mid (t1,i1) \in TI \wedge (t1,t2) \in CT :$$
$$\sum_{rt \in RT} rtt_{rt,t2} \geq cp_{t1,t2} + mi_{t1,i1} + \sum_{\substack{(t2,i2) \in TI \mid \\ i2 \in I^h \wedge i1 \neq i2}} mi_{t2,i2} - 2$$
$$\forall t \in T :$$
$$\sum_{rt \in RT} rtt_{rt,t} \geq \sum_{(t2,t) \in CP : (t,t2) \notin CT} cp_{t2,t} \tag{6}$$

- guarantees that a reconfiguration between tasks $t1 \in T$ and $t2 \in T$ is executed after $t1$ and before $t2$ (T_{max} identifies the maximum possible execution time for the schedule)
$$\begin{aligned} &\forall rt \in RT, \forall t \in T : \\ &\quad b_t \geq e_{rt} - (1 - rtt_{rt,t}) \cdot T_{max} \\ &\forall rt \in RT, \forall (t,t2) \in CP : \\ &\quad e_t \leq b_{rt} + (2 - rtt_{rt,t2} - cp_{t,t2}) \cdot T_{max} \end{aligned} \tag{7}$$

Notice that the proposed model currently does not directly consider the delay due to communication among tasks. However, Equation 1 can be easily modified to take into account a fixed $\lambda_{t1,t2}$ communication time among tasks $t1$ and $t2$ that is not dependent on the selected implementations:

$$\forall (t1,t2) \in P : b_{t2} \geq e_{t1} + \lambda_{t1,t2} \tag{8}$$

Starting from the variables and parameters previously defined, it is possible to compute and thus optimize the following three different metrics:

Makespan (T_{cost}): the overall execution time needed to complete the computation of all the tasks of the schedule including reconfiguration tasks;

Peak power (P_{cost}): the estimated power reached by the schedule, computed considering the maximum overall power consumption reached within a single time unit;

Energy consumption (E_{cost}): the estimated energy consumption for the schedule, it is computed considering the specific implementation selected for each task and the energy needed for all the reconfigurations.

Overall, a possible objective function for the problem can be obtained with a linear combination of T_{cost}, P_{cost} and E_{cost}:

$$\min \left\{ q_1 \cdot \frac{T_{cost}}{T_{max}} + q_2 \cdot \frac{P_{cost}}{P_{max}} + q_3 \cdot \frac{E_{cost}}{E_{max}} \right\} \tag{9}$$

Where T_{max}, P_{max} and E_{max} are normalization terms representing the maximum value that T_{cost}, P_{cost} and E_{cost} can achieve respectively, while q_1, q_2 and q_3 are weights that can be set according to the designer preferences.

V. Iterative scheduling algorithm

Even though the model in the previous section could be solved directly by a MILP solver to achieve the optimal solution, this approach can only be adopted for taskgraphs with a small number of nodes. Indeed, the size of the model increases quadratically with respect to the number of tasks, so that the resulting formulation becomes challenging to solve. In order to overcome this issue, we consider an iterative approach that schedules a subset of the tasks per iteration reducing the size of the formulation. For each iteration the solver computes the optimal solution, with respect to the objective function provided by the designer, for the reduced formulation; then, this solution is forced into the following iteration setting some binary variables of the MILP model ($mic_{t,i,c}, cp_{t1,t2}, cft_{c,t}, rtt_{rt,t}$) reducing the solution space even further. Forcing the $mic_{t,i,c}$ variables means that in the following iteration the tasks t that have been already scheduled can not change their implementation i or component c where they are mapped on. Forcing the $cp_{t1,t2}$ variables avoids that two tasks $t1$ and $t2$ mapped on the same hardware component change their scheduled order, even if they are not constrained by a precedence relation in the taskgraph. Forcing the $cft_{c,t}$ variables means that the first task t on a hardware component c can not be changed. Finally, forcing the $rtt_{rt,t}$ variables ensures the reconfiguration rt for the scheduled tasks t. Notice however that the values of the beginning b_t and the end e_t variables of the tasks are not forced, this leaves some flexibility to perform backtracking of previous decisions. The pseudo code of the iterative scheduler is shown in Algorithm 1.

```
input : mod: the MILP model
        k: number of tasks to consider at each iteration
output: solution of the MILP model
taskQueue ← topologicalSort(mod.T, mod.P)
mod.T ← ∅
while taskQueue is not empty do
    for i ← 1 to k do
        mod.T = mod.T ∪ taskQueue.pop()
    end
    sol ← computeMILPSolution(mod)
    mod.mic_{t,i,c} ← sol.mic_{t,i,c}
    mod.cp_{t1,t2} ← sol.cp_{t1,t2}
    mod.cft_{c,t} ← sol.cft_{c,t}
    mod.rtt_{rt,t} ← sol.rtt_{rt,t}
end
return sol
```

Algorithm 1: Iterative scheduling algorithm

In Figure 1 are shown the iterations of our iterative scheduler on a 7 tasks application scheduling $k = 3$ tasks at the time. In the first iteration the optimal solution for the first 3 tasks is found and passed to the following iteration setting those already discussed binary variables. In the second iteration the next 3 tasks are scheduled without additional constraints, while the previous 3 are forced in the current solution. Eventually, in the third iteration, the last task of the application is scheduled, while the previous 6 are forced in the final schedule.

VI. Experimental evaluation

In this section we present an evaluation of the iterative scheduling algorithm comparing the achieved results with respect to the ACO-based algorithm [10] that we enhanced adding the evaluation of the peak power and energy consumption metrics. The iterative scheduler was implemented in python using Gurobi 6.0 [15] for the optimization of the MILP model, while the experiments were performed on a Intel Core i7-2630QM under linux.

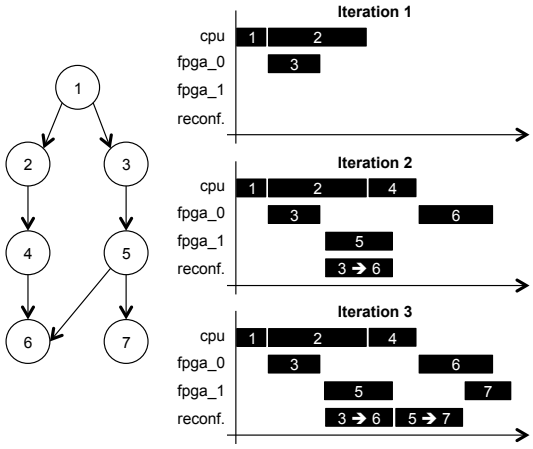

Fig. 1. Iterative scheduler example execution with $k = 3$ (IS-3).

In order to evaluate the effectiveness of our approach we considered a case study consisting of real tasks in the context of image analysis. The task graph of the application is shown in Figure 2 and it consists of two main computation chains. The first chain includes the tasks: histogram, Otsu filter and

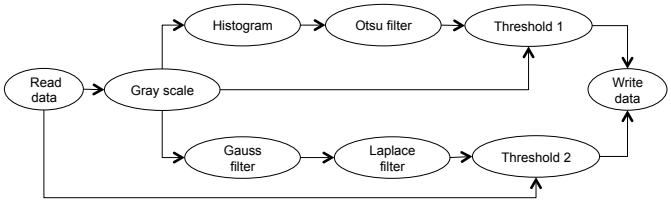

Fig. 2. Taskgraph of the image analysis case study.

threshold 1, the goal of this chain is to binarize the image applying the Otsu separation algorithm [16]. The second chain consists of the tasks: Gauss filter, Laplace filter and threshold 2, the purpose of this part of the computation is to perform edge detection within the image. For both the computations the image needs to be gray scaled while the final results are stored in main memory.

The target architecture considered for our application is the ZedBoard with Zynq^{TM}-7000 AP SoC that provides a dual core ARM Cortex-A9 CPU and a Xilinx XC7Z020 FPGA. For each of the tasks (except for the initial read data and final write data) we generated both hardware and software implementations. Multiple hardware implementations were generated using Vivado HLS with and without loop unrolling, moreover each hardware core was placed and routed separately in order to obtain performance and power consumption estimates. The execution time of the hardware implementations was estimated considering the HLS reports together with the clock frequency for the design, on the other hand, the execution time of the software implementations was measured directly on the CPU. The communication between the tasks is accomplished using DMA and the communication overhead is added directly to the estimated execution time of each task.

Since the number of tasks involved in the application is manageable, we performed a full search of the solution space by executing the iterative scheduler with $k = 9$, while the execution time of the MILP solver was limited to 100 seconds. On the other hand, 10 different executions of the ACO-based algorithm [10] were executed and the best solution was considered for comparison. The weights of the objective function were set to $q_1 = 0.5$, $q_2 = 0$, $q_3 = 0.5$ so that execution time and energy consumption were taken into account to the same extent. The solutions obtained by [10] and the proposed approach are shown in Figure 3 and 4 respectively. The y axis shows the software (arm0, arm1) and hardware

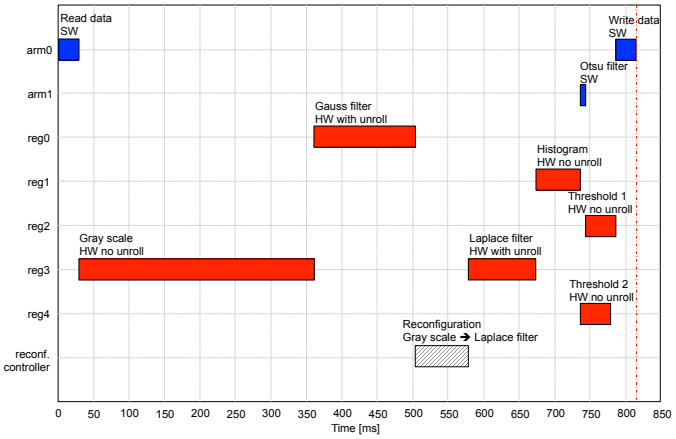

Fig. 3. Solution obtained by [10] on the image analysis case study.

(reg0, reg1, ...) components together with the reconfiguration controller, while the execution time of the tasks is represented on x axis. For the tasks implemented on the reconfigurable regions the figures also report if loop unrolling was used or not. Overall the iterative algorithm achieved a schedule

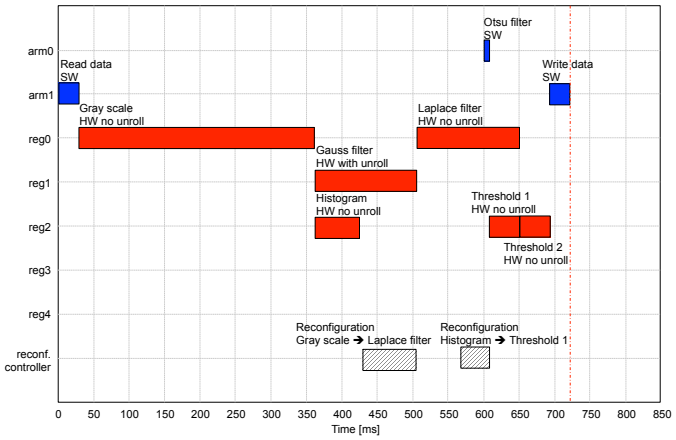

Fig. 4. Solution obtained by the iterative scheduler on the image analysis case study.

execution time of 721ms with an energy consumption of 1221mJ, while the schedule obtained by [10] requires 817ms for completion and has an energy consumption of 1512mJ. The 11.8% reduction in the execution time is mainly achieved by exploiting reconfiguration prefetching of the Laplace filter task. On the other hand the 19.2% energy saving is due to both the reduction of static power consumption and to the selection of implementations that have higher energy efficiency. Notice

TABLE III
SCHEDULE MAKESPAN AND ALGORITHM EXECUTION TIME COMPARISON

# Tasks	Average schedule makespan [ms]						Average execution time [s]					
	[10]	IS-1	IS-2	IS-3	IS-4	IS-5	[10]	IS-1	IS-2	IS-3	IS-4	IS-5
10	536	450	511	481	477	468	9.23	0.95	0.76	0.84	0.94	34.71
20	1266	1082	987	928	819	845	28.36	5.86	4.28	5.66	37.51	749.58
30	2272	1676	1516	1379	1351	1168	52.14	26.02	17.32	17.61	79.75	764.16
40	6216	5002	4641	4802	4760	4527	76.08	36.86	21.85	21.08	47.12	393.79
50	3984	2876	2950	2861	2515	2219	119.4	115.6	71.01	61.4	151.74	1079.34

also that the iterative scheduler exploits module reuse for tasks threshold 1 and threshold 2 that perform the same functionality. This leads to a reduced number of reconfigurable regions and a corresponding reduction of resources required on the FPGA. For a broader validation of our approach, we also performed some tests on a set of 36 pseudo-random generated taskgraphs having different numbers of nodes in the range $[10, 50]$. For each task we generated, on average, 4 different hardware/software implementations ensuring the presence of common functionalities to allow the exploitation of module reuse. Furthermore, the implementations have been generated taking into account that generally larger resource requirements are correlated to a smaller execution time. For each problem instance we executed the iterative scheduler (IS-k) varying the number of tasks per iteration (k from 1 to 5) and we compared the results of our approach with respect to [10], considering both the schedule makespan and the energy consumption in different test sessions. Table IV summarize the improvements obtained by IS-k over the ACO-based algorithm, while Table III reports detailed information on the makespan benchmark. It is worth noting that IS-4 gives an average reduction of the makespan of 23.7% with an execution time that is comparable to [10]. On the other hand, even though IS-5 leads to higher quality solutions, the time needed to solve the MILP model is in general significantly higher.

TABLE IV
AVERAGE SCHEDULE IMPROVEMENTS W.R.T. [10]

metrics	IS-1	IS-2	IS-3	IS-4	IS-5
makespan	16.0%	19.0%	21.4%	23.7%	27.7%
energy consumption	38.2%	38.6%	37.8%	38.1%	38.3%

VII. CONCLUSIONS

We introduced a novel MILP formulation for scheduling applications on heterogeneous architectures, which takes advantage of PDR, module reuse and reconfiguration prefetching in order to improve the returned schedule. Furthermore, we shown that our approach is able to improve the quality of state-of-the-art algorithms within a comparable execution time.

Future work we will investigate the possibility of reducing the size of the MILP model in terms of number of variables and constraints. This could lead to a faster exploration of the solution space and the opportunity to apply the iterative scheduler using larger subset of tasks. We will also consider to extend the model to take into account different kind of communications among tasks on heterogeneous architectures.

REFERENCES

[1] B. L. Hutchings and M. J. Wirthlin, "Implementation approaches for reconfigurable logic applications," in *Field-Programmable Logic and Applications*. Springer, 1995, pp. 419–428.

[2] F. Redaelli, M. D. Santambrogio, and S. O. Memik, "An ilp formulation for the task graph scheduling problem tailored to bi-dimensional reconfigurable architectures," *Int. J. Reconfig. Comput.*, vol. 2009, pp. 7:1–7:12, Jan. 2009.

[3] S. Fekete, E. Kohler, and J. Teich, "Optimal fpga module placement with temporal precedence constraints," in *Design, Automation and Test in Europe, 2001. Conference and Exhibition 2001. Proceedings*, 2001, pp. 658–665.

[4] S. Banerjee, E. Bozorgzadeh, and N. Dutt, "Integrating physical constraints in hw-sw partitioning for architectures with partial dynamic reconfiguration," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 14, no. 11, pp. 1189–1202, Nov 2006.

[5] Y. M. Lam, J. Coutinho, W. Luk, and P. H. W. Leong, "Mapping and scheduling with task clustering for heterogeneous computing systems," in *Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on*. IEEE, 2008, pp. 275–280.

[6] S. Banerjee, E. Bozorgzadeh, and N. Dutt, "PARLGRAN: parallelism granularity selection for scheduling task chains on dynamically reconfigurable architectures," in *Proceedings of the 2006 Conference on Asia South Pacific Design Automation: ASP-DAC 2006, Yokohama, Japan, January 24-27, 2006*, 2006, pp. 491–496.

[7] B. Mei, P. Schaumont, and S. Vernalde, "A hardware-software partitioning and scheduling algorithm for dynamically reconfigurable embedded systems," in *Proceedings of ProRISC*. Citeseer, 2000, pp. 405–411.

[8] S. Ghiasi, A. Nahapetian, and M. Sarrafzadeh, "An optimal algorithm for minimizing run-time reconfiguration delay," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 3, no. 2, pp. 237–256, 2004.

[9] P. Brucker, A. Drexl, R. Möhring, K. Neumann, and E. Pesch, "Resource-constrained project scheduling: Notation, classification, models, and methods," *European journal of operational research*, vol. 112, no. 1, pp. 3–41, 1999.

[10] R. Cattaneo, R. Bellini, G. Durelli, C. Pilato, M. Santambrogio, and D. Sciuto, "Para-sched: A reconfiguration-aware scheduler for reconfigurable architectures," in *Parallel Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International*, May 2014, pp. 243–250.

[11] J. Henkel, "A low power hardware/software partitioning approach for core-based embedded systems," in *Proceedings of the 36th Annual ACM/IEEE Design Automation Conference*, ser. DAC '99. New York, NY, USA: ACM, 1999, pp. 122–127.

[12] M. Goraczko, J. Liu, D. Lymberopoulos, S. Matic, B. Priyantha, and F. Zhao, "Energy-optimal software partitioning in heterogeneous multiprocessor embedded systems," in *Proceedings of the 45th annual design automation conference*. ACM, 2008, pp. 191–196.

[13] M. Rabozzi, A. Miele, and M. D. Santambrogio, "Floorplanning for partially-reconfigurable FPGAs via feasible placements detection," in *Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23nd Annual International Symposium*, 2015, pp. 252–255.

[14] Xilinx Inc, "Vivado Design Suite User Guide: Partial Reconfiguration," 2014.

[15] I. Gurobi Optimization, "Gurobi optimizer reference manual," 2015. [Online]. Available: http://www.gurobi.com

[16] N. Otsu, "A threshold selection method from gray-level histograms," *Automatica*, vol. 11, no. 285-296, pp. 23–27, 1975.