Task Scheduling with Configuration Prefetching and Anti-Fragmentation techniques on Dynamically Reconfigurable Systems

Francesco Redaelli, Marco D. Santambrogio, Donatella Sciuto

{santambr,sciuto}@elet.polimi.it, francesco.redaelli@dresd.org

ABSTRACT

Aim of this paper is to define a scheduling of the task graph of an application that minimizes its total execution time on a partially dynamically reconfigurable FPGA. The scheduler has to take into account the reconfiguration overhead of each task, the area constraint of the target FPGA, the precedences between the tasks, *configuration prefetching* and *module reuse*. We introduce an ILP formulation to solve the task scheduling problem in the reconfigurable architecture scenario. This formulation has been used to identify interesting features for a possible heuristic scheduler. The results of the ILP solution show how a reconfiguration-aware scheduler exploiting all the reconfiguration features can outperform one with partial knowledge.

1. INTRODUCTION

Reconfigurable hardware in general, and FPGAs in particular, have received much attention over the last years. At first they have been employed as a cheap means of prototyping and testing hardware solutions without having to undergo the long and expensive process of ASIC design, thus allowing to drastically reduce the time-to-market. FPGAs have been so successful in this task that nowadays it is not uncommon to even directly *deploy* FPGA–based solutions. In this scenario, that can be termed *Compile Time Reconfiguration* (CTR), the configuration of the FPGA is loaded at the end of the design phase, and it remains the same throughout the whole time the application is running. In order to change the configuration one has to stop the computation, reconfigure the chip resetting it, and then start the new application. CTR was for some years the only kind of reconfiguration available for FPGAs. With the evolution of technology, though, it became possible to considerably reduce the time needed for the chip reconfiguration: this made it conceivable to reconfigure the FPGA *between* different stages of its computation, since the induced time overhead could be considered acceptable. This process is called *Run Time Reconfiguration* (RTR), and the FPGA is said to be *Dynamically Reconfigurable*. RTR can be exploited by creating what has been termed *virtual hardware* [1, 2] in analogy with the concept of *virtual memory* in general computers. Consider an application that is too big to fit into a particular FPGA: one can *partition* it into n smaller tasks, each one fitting on the chip. Then it is possible to load task 1 on the chip, execute it, then reconfigure the FPGA for task 2 and execute it, and so on until task n is finished. This idea is called *time partitioning*, and has been studied extensively in literature (see [3–6]). A further improvement in FPGA technology allows modern boards to reconfigure only *some* of the logic gates, leaving the other ones unchanged. This *partial reconfiguration* is of course much faster in case only a small part of the FPGA logic needs to be changed. When both these features are available, the FPGA is called *partially dynamically reconfigurable*. This can be done using partial reconfiguration bitstreams[1]. The main characteristic of bitstreams is that they have a correlation with the operation they are implementing: once the bitstream is defined the operation is defined too, while given an operation, there could exist more than one bitstream implementing it. Therefore it is possible to assign to each bitstream an attribute called *type* used to identify the operation implemented, the area occupied on the target architecture and the time needed to be configured and to be executed by that bitstreams.

Aim of this work is to propose an exact ILP formulation for the task scheduling problem in a reconfigurable scenario. Let us define a set of reconfiguration features that have to be taken into account to define the schedule. *Configuration prefetching* means that a module is loaded onto the FPGA as soon as possible in order to hide its reconfiguration time as much as possible. *Module reuse* means that two tasks of the same type have the possibility to be executed exactly on the same module on board, hiding completely the reconfiguration time. *Configuration prefetching* and *module reuse* are combined with time partitioning techniques to optimize the latency of the application [7] [8] [9]. *Anti-fragmentation techniques* avoid the fragmentation of the available space on board trying to maximize the dimension of free adjacent areas. The *deconfiguration policy* is a set of rules used to decide when and how to remove a module from the FPGA. The task scheduling problem, working with a partial dynamic reconfigurable architecture is similar to the one proposed in [10] and [11]. In [10] and [11] the authors consider the problem of partitioning and scheduling a task dependence graph onto an architecture defined using a general purpose processor (GPP) and reconfigurable devices where tasks can be executed both as software onto the GPP or as hardware cores using the reconfigurable resources. They propose a heuristic HW/SW partitioning and scheduling algorithm based on the well known KLFM heuristic. The problem with these works is that they do not consider all the features available to partially dynamically reconfigurable devices i.e., *module reuse* is rarely considered, and *anti-fragmentation techniques* are almost always ignored. Our approach models the scheduling problem through an ILP formulation that takes into account all specific features of partially dynamically reconfigurable architectures.

[1] a bitstream is a binary file used to configure an FPGA, or part of it, with the desired functionality.

Section 2 describes the state of the art, proposing solutions to related problems. Section 3 introduces the ILP model. Section 4 presents a set of experimental results obtained by running the developed scheduler and it compares our ILP formulation with the one proposed in [11]. Finally, Section 5 presents the conclusions of this work.

2. PROBLEM DESCRIPTION

Main goal of this work is to be able to execute onto a dynamically partially reconfigurable FPGA, an application that is too large to be completely mapped onto that FPGA. Thus, the application has to be partitioned in such a way that allows the mapping onto the target hardware using its partial reconfiguration capability. The aim is to minimize the total latency of the partitioned application. The formal description of the problem is the following. Let us consider the behavioral description of the application in terms of a DAG task graph, in which each node describes the task operations. Each task node is associated with a bitstream and therefore its execution time, the amount of area used on the target FPGA and the reconfiguration time needed. Given this information the aim is to define a scheduling of the task graph that minimizes the total execution time of the algorithm; the scheduler has to take into account the reconfiguration overhead of each task, the area constraint of the target FPGA, the precedences between the tasks, *configuration prefetching* and *module reuse*. The scheduler has to give a solution in terms of:

- when to reconfigure a module and for which task;
- where to place the reconfigured module on the FPGA;
- when to start the execution of a task according with its precedence constraints.

The FPGA architectures exploiting the partial dynamic reconfiguration feature may be seen as a sequence of reconfigurable columns, where a column is the minimum amount of reconfigurable area that can be used. In this scenario, the area property of a task can be expressed as the number of adjacent columns needed by the task on the FPGA.

2.1 Related work

In [8] the authors present an ILP formulation considering a task graph, partitioned into time–partitions, defined only with few task types achieving results similar to [12]. One of the first attempts to take into consideration partial reconfiguration combined with *configuration prefetching* is presented in [13]. The applications that can be scheduled with this approach are linear multi-task applications: they are a linear sequence of tasks and each task takes as input the results of the previous task; furthermore this kind of applications normally deals with large amounts of data e.g., image processing. The goal in [13] is to define a specific methodology for scheduling the tasks of these applications in order to reduce the overall completion time. The same authors present in [7] an enhanced solution for the same problem: *PARLGRAN* tries to reduce the total execution time using two different techniques. The former one is called *simple fragmentation reduction* and it places the new task in the first available area on the FPGA in the opposite side of the FPGA with respect to the location of the previous task. The latter is called *exploiting slack in reconfiguration controller* and it is a local optimization that postpones the reconfiguration of a task in a position selected by the previous technique if this reconfiguration causes a delay in the subsequent task execution time. Furthermore, a task is replicated if and only if the replication gives an improvement in the execution time of the subsequent task considered as unreplicable; this technique is called *static pruning*. For replicated tasks there is a specific placement policy called *dynamic granularity selection*: placing multiple copies of a task in adjacent positions can reduce the fragmentation of the area of the FPGA; moreover, if it leads to a temporal improvement, the first copies of a task are stopped early in order to overlap the reconfiguration of the following task with the final execution of the last copies of the considered one. The authors do not consider the memory management in a context of parallel tasks that work on different portions of the same data; moreover the memory available for the tasks is not quantified and this could be a problem. In [9] a reconfiguration sequence manager is discussed, using as target architecture a partially dynamically reconfigurable system (partially dynamically reconfigurable FPGA systems or multi-FPGA systems). The proposed algorithm receives as input an already scheduled task graph and it aims at scheduling all these tasks onto the target architecture in order to minimize only the total reconfiguration time needed. The assumptions are the following: (a) every task has a fixed size equal for each task, with the same reconfiguration time; (b) on the target architecture there are K fixed positions where to put a task.

Under these assumptions the algorithm finds a provable optimal solution [9], and the techniques used are based on the *off-line paging* solved by *Least Imminently Used (LIU)* algorithm. This approach does not take into consideration resource reuse and different task sizes. Furthermore, only the total reconfiguration time is minimized, without considering the execution time and the reconfiguration time impact described for example in [14] for a similar architecture.

3. ILP FORMULATION

Let us introduce the ILP formulation used to solve the task scheduling problem in the reconfigurable architecture scenario.

3.0.1 Constants

For every task $i \in O$ let's us define

- l_i := latency of i;
- d_i := reconfiguration time needed by i;
- r_i := room occupied onto the FPGA by i, expressed in number of (adjacent) columns of the FPGA.

Moreover, using the operator $[\cdot]$ that takes value 1 if $\cdot$ is true and 0 otherwise, let:

- a_{ij} := [tasks i and j perform the same action[2]].

Also, it is necessary to consider at most T time instants, where

$$T := \sum_{i=1}^{|O|} (l_i + d_i). \quad (1)$$

The FPGA is composed of $|U|$ columns.

[2]i.e. they can exploit reuse.

3.0.2 Variables

The following variables are defined:

- p_{ihk} := [task i is present on the FPGA at time h and the leftmost column it uses is the k-th];
- $\bar{t}_{ih}$:= [the reconfiguration of task i starts at time h];
- m_i := [task i exploits module reuse];
- S_i^{on} := arrival time of task i on the FPGA;
- S_i^{off} := last time instant when task i is on the FPGA;
- t_{e} := overall execution time.

3.0.3 Constraints

The constraints marked with a * are written with the if–then transformation (see [15]).

Area constraints No more than $|U|$ columns can be used at any time:

$$\forall h, \sum_{i=1}^{|O|} \sum_{k=1}^{|U|} p_{ihk} \cdot r_i \leq |U|, \quad \text{Area Constraint} \tag{2}$$

Also, a column can't be used by more than one task at the same time:

$$\forall h, k, \sum_{i=1}^{|O|} \sum_{l=\max(k-r_i+1,1)}^{k} p_{ihl} \leq 1, \quad \text{Non Overlap} \tag{3}$$

The $c_i - 1$ columns to the right of the leftmost column of a task i can't be used by another task since they are needed by i:

$$\sum_{i=1}^{|O|} \sum_{h=1}^{T} \sum_{k=|U|-r_i+2}^{|U|} p_{ihk} = 0, \quad \text{Right space} \tag{4}$$

Timing constraints The starting instant is reserved, see (13):

$$\sum_{i=1}^{|O|} \sum_{k=1}^{|U|} p_{i0k} = 0, \quad \text{Zero time} \tag{5}$$

The 1's in p_{ihk} are arranged in a column for every task i for the time it is on the FPGA:*

$$\forall i, h, k, \sum_{m=1}^{T} \left(\sum_{l=1}^{|U|} p_{iml} - p_{imk} \right) \leq T(1 - p_{ihk}), \quad \text{Same column} \tag{6}$$

Arrival time must be smaller than or equal to the first instant for which p is 1:*

$$\forall i, h, S_i^{\text{on}} - h \sum_{k=1}^{|U|} p_{ihk} \leq T \left(1 - \sum_{k=1}^{|U|} p_{ihk} \right), \quad \text{Arrival time} \tag{7}$$

Leaving time must be greater than or equal to the last instant for which p is 1:

$$\forall i, h, S_i^{\text{off}} \geq h \sum_{k=1}^{|U|} p_{ihk}, \quad \text{Leaving time} \tag{8}$$

The tasks cannot disappear and reappear from the FPGA:

$$\forall i, \sum_{h=1}^{T} \sum_{k=1}^{|U|} p_{ihk} = S_i^{\text{off}} - S_i^{\text{on}} + 1, \quad \text{Continuous usage} \tag{9}$$

We must enforce precedences:

$$\forall (i,j) \in P, S_j^{\text{off}} - l_j \geq S_i^{\text{off}}, \quad \text{Precedences} \tag{10}$$

Reconfiguration constraints If a task does not exploit reuse, then it must be on the FPGA for at least[3] the time necessary for reconfiguration and execution:*

$$\forall i, d_i + l_i - \sum_{h=1}^{T} \sum_{k=1}^{|U|} p_{ihk} \leq T \cdot m_i, \quad \text{Reconfigured time} \tag{11}$$

Reconfiguration starts as soon as the task is on the FPGA:*

$$\forall i, S_i^{\text{on}} - \sum_{h=1}^{T} h \cdot \bar{t}_{ih} = T \cdot m_i, \quad \text{Rec. start}$$

Only one reconfiguration can take place at a time:

$$\forall h, \sum_{i=1}^{|O|} \sum_{m=\max(1,h-t_i^r+1)}^{h} \bar{t}_{im} \leq 1, \quad \text{Single rec.} \tag{12}$$

A task can exploit module reuse at time h if and only if at time $h-1$ there is in the same position the same task or an equivalent task:*

$$\begin{aligned} \forall i, \forall h, \forall k, 1 - \sum_{j=1, j\neq i}^{|O|} a_{ij} \cdot p_{j(h-1)k} - p_{i(h-1)k} + \\ -T \cdot (1 - p_{ihk}) \leq T \cdot (1 - m_i), \quad \text{Reuse} \end{aligned} \tag{13}$$

If a task does exploit reconfiguration, it has to remain on the FPGA at least for the execution time:*

$$\forall i, l_i - \sum_{h=1}^{T} \sum_{k=1}^{|U|} p_{ihk} \leq T(1 - m_i), \quad \text{Reused time} \tag{14}$$

The reconfiguration time is unique. If there is task reuse, it is not defined:

$$\forall i, \sum_{h=1}^{T} \bar{t}_{ih} \leq 1 - m_i, \quad \text{No reconfiguration} \tag{15}$$

Definition of t_{e}

$$\forall i, t_{\text{e}} \geq S_i^{\text{off}}, \qquad t_{\text{e}} \leq T \tag{16}$$

3.0.4 Objective

$$\min t_{\text{e}} \tag{17}$$

4. EXPERIMENTAL RESULTS

To show the importance of a reconfiguration aware scheduler, the solutions obtained solving the ILP proposed in Section 3 are compared with the ones obtained solving the modified ILP formulation presented in [11]. In [11] Dutt presents an ILP model that describes a scheduler for *HW/SW-Codesign* based system, thus, it needs to be modified to allow a scheduling for only dynamically partially reconfigurable hardware

[3]Reconfiguration prefetching is allowed

Table 1: ILP results comparison

Task Graph	TG1	TG2	TG3	TG4	TG5	TG6	TG7	TG8	TG9	TG10
Proposed ILP	17	22	16	15	21	22	17	18	26	23
Dutt ILP	21	25	22	25	28	22	24	23	31	32

architectures, in particular FPGAs. This is done by forcing all the tasks to be executed in hardware. This new ILP exploits the *configuration prefetching* techniques, but does not consider the *module reuse* concepts. It is simple to understand that the ILP proposed in Section 3 will obtain results in terms of schedule length better or at least equal to those obtained with the Dutt ILP formulation. Table 1 presents results obtained scheduling ten different task graphs with the two ILP formulations.

The task graphs used to obtain these results have been designed manually, and each one has ten tasks. To make the ILPs feasible to be solved, the maximum execution time for a task has been set to 5, while the maximum reconfiguration time to 3; furthermore, the FPGA where the tasks are scheduled has only 5 columns (except for *Ten7* and *Ten8* where the columns are 6). This kind of graphs does not represent any realistic specification, but are useful to understand how *module reuse* can affect the schedule length. Table 1 shows that the proposed ILP formulation obtains results sometime much better than the Dutt one, but in some cases this gap can be small or even null: in *Ten6*, for instance, even using *module reuse* the task graph characteristics do not allow for a better schedule.

The examples proposed in this section consider only very small task graphs and FPGAs: this is done to limit the execution time of the ILP solver. Even for these graphs the solver needs several days of work: this is a proof of the unfeasibility of using an ILP scheduler for real applications, and the need to develop heuristic approaches that can however benefit from the ILP model for their definition.

5. CONCLUSIONS AND FUTURE WORK

The aim of this work was to develop an ILP model for the problem of scheduling a task graph on a partially dynamically reconfigurable architecture. This model takes into account all the possible features exploitable on this kind of architectures: *configuration prefetching*, *module reuse* and anti-fragmentation techniques. The results show how the joined use of all these techniques can considerably improve the schedule length.

6. REFERENCES

[1] Xiao ping Ling and Hideharu Amano. Performance evaluation of wasmii: a data driven computer on a virtual hardware. In Arndt Bode, Mike Reeve, and Gottfried Wolf, editors, *PARLE*, volume 694 of *Lecture Notes in Computer Science*, pages 610–621. Springer, 1993.

[2] William Fornaciari and Vincenzo Piuri. Virtual fpgas: Some steps behind the physical barriers. In *IPPS/SPDP Workshops*, pages 7–12, 1998.

[3] João M. P. Cardoso. On combining temporal partitioning and sharing of functional units in compilation for reconfigurable architectures. *IEEE Trans. Computers*, 52(10):1362–1375, 2003.

[4] João M. P. Cardoso and Horácio C. Neto. Compilation for fpga-based reconfigurable hardware. *IEEE Design & Test of Computers*, 20(2):65–75, 2003.

[5] Meenakshi Kaul, Ranga Vemuri, Sriram Govindarqjan, and Iyad Ouaiss. An automated temporal partitioning and loop fission approach for fpga based reconfigurable synthesis of dsp applications. In *DAC '99: Proceedings of the 36th Annual Conference on Design Automation (DAC'99)*, pages 616–622. IEEE Computer Society, 1999.

[6] Joo M. P. Cardoso. Loop dissevering: A technique for temporally partitioning loops in dynamically reconfigurable computing platforms. In *IPDPS '03: Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03)*, page 181.2. IEEE Computer Society, 2003.

[7] S. Banerjee, E. Bozorgzadeh, and N. Dutt. Parlgran: parallelism granularity selection for scheduling task chains on dynamically reconfigurable architectures. *In Proceedings of Asia and South Pacific Design Automation Conference, ASP_DAC*, January 2006.

[8] K. Jafri, N. Jafri, and S. Khan. Constraint based temporal partitioning model for partial reconfigurable architectures. *In Proceedings of IEEE INIMIC*, pages 242–246, 2003.

[9] S. Ghiasi and M. Sarrafzadeh. Optimal reconfiguration sequence management. *In Proceedings of Asia South Pacific Design Automation Conference, ASP_DAC*, pages 359–365, 2003.

[10] S. Banerjee, E. Bozorgzadeh, and N. Dutt. Physically-aware hw-sw partitioning for reconfigurable architectures with partial dynamic reconfiguration. *In DAC 2005*, pages 335–340, June 2005.

[11] S. Banerjee, E. Bozorgzadeh, and N. Dutt. Integrating physical constraints in hw-sw partitioning for architectures with partial dynamic reconfiguration. *In IEEE Transaction on very large scale integration systems*, 14(11):1189–1202, November 2006.

[12] M. Kaul and R. Vemuri. Temporal partitioning combined with design space exploration for latency minimization of run-time reconfigured design. *In Proceedings of Design, Automation and Test in Europe Conference and Exhibition*, pages 202–209, DATE 1999.

[13] S. Banerjee, E. Bozorgzadeh, and N. Dutt. Considering run-time reconfiguration overhead in task graph transformation for dynamically reconfigurable architectures. *In Proceedings of the 13th annuals IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM'05*, 2005.

[14] J. Resano et al. Specific scheduling support to minimize the reconfiguration overhead of dynamically reconfigurable hardware. *In DAC'04*, June 2004.

[15] Wayne L. Winston. *Introduction to Mathematical Programming: Applications and Algorithms*. Duxbury Resource Center, 2003.