2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines

A Communication Aware Online Task Scheduling Algorithm for FPGA-based Partially Reconfigurable Systems

Yi Lu, Thomas Marconi, Koen Bertels and Georgi Gaydadjiev
Computer Engineering Lab., TU Delft, The Netherlands
{$Yi.Lu, T.M.Thomas, k.l.m.bertels, g.n.gaydadjiev$}@tudelft.nl

Abstract

In this paper, we propose an efficient online task scheduling algorithm which targets 2D FPGA area partitioning model and takes into account the data dependency and the data communications 1) among hardware tasks and 2) between hardware tasks and external devices which have not been explicitly investigated in previous work. In the experiment with 10000 workloads, the evaluation result shows that our proposed scheduling algorithm is about 20x faster than the comparable approach.

I. Introduction

For FPGA-based partially reconfigurable systems, one of the key challenges is to provide the appropriate runtime management of the configurable resources which is referred to as reconfigurable hardware operating system (**RHOS**). The RHOS manages all configurable resources to avoid any conflict among various configurations as well as to achieve efficient resource usage. Two critical parts of the RHOS are the task placement and task scheduler which determine the allocation and the starting and ending times of hardware tasks running on a FPGA respectively.

Nowadays, the 2D area partitioning model of FPGAs (**2D area**) [1] is broadly used. The loading and removing of hardware tasks impact the free resources on the FPGA. So a successful task scheduling decision means that a task is assigned to some configurable resources (spatial requirements) which will not be used by any other tasks during its scheduled time (temporal requirements).

Previously proposed 2D area task scheduling algorithms showed the spatial requirement (e.g.[2][3]), as well as parts of the temporal requirements (configuration and execution times, e.g. [4][5]). However, there is less investigation into data communications among tasks and external devices which introduce another critical temporal requirement: tasks' data communication time. Without taking the communication requirement into account, the scheduler may not guarantee correct data exchanges among scheduled tasks and external devices during an application's execution.

In this paper, we propose an efficient 2D area task scheduling algorithm supporting these communication constraints. The main contributions of this paper are:

- the definition of an efficient communication aware task scheduling algorithm based on our 2D area model and communication infrastructure;
- elaboration and evaluation of a scheduling heuristic;

The remainder of the paper is structured as follows: in section II, related work is presented. Then, in section III, we define the models used in our scheduling algorithm and analyze the common problems of scheduling tasks in the partially reconfigurable systems. Thereafter, we detail our algorithm in section IV. In section V, we present the experimental results. Finally, we conclude this paper and discuss future work.

II. Related work

Steiger et al. [2] converted the online 1D area scheduling problem to a strip packing problem. In addition, they proposed and evaluated the "1D Horizon" and "Stuffing" scheduling heuristics. Banerjee et al. [3] also applied the strip packing model into the PARLGRAN which is an application mapping and scheduling approach.

Zhou et al. [6] proposed a compact reservation (**CR**) scheduling algorithm which modeled the FPGA area as a 2D matrix. For each arriving task, the algorithm recalculates all the encoded information stored in matrix units to find all available allocations.

Fu et al. [4] proposed a knapsack algorithm based scheduler where the application execution time is divided into continuous time intervals. In the beginning of each time interval, their multi-constraint knapsack heuristic chooses the tasks that need to be mapped on the FPGA.

None of these approaches treat the communication time as a specific constraint when scheduling tasks. Our proposed online task scheduling algorithm applies the strip packing to the 2D area as well as takes into account the task communication time as a specific constraint.

III. Models and problem analysis

In this section, we first detail our system model, the 2D area with communication infrastructure, and the task model which will be used in our scheduling algorithm. Then based on these models, we analyze the common problems for scheduling a task into the FPGA-based partially reconfigurable systems and present possible solutions.

A. Models

*1) **System model**:* Figure 1 shows the target system where the partial reconfiguration on the FPGA is controlled by the scheduler and bitstream loader running on the host processor. When a task arrives, the scheduler allocates the task by taking into account the start and end time. According to the scheduler's decisions, the bitstream loader configures the required tasks at their scheduled times. The tasks running on the FPGA can communicate not only with the host processor via an exchange memory, but can also read (write) data from (to) the peripherals connected to the FPGA.

*2) **2D area and its communication infrastructure**:* The FPGA area is first partitioned into slots, then each slot is further partitioned into blocks. As shown in Figure 2, such a block is referred to as a **configuration block** which is defined as the minimum configuration

978-0-7695-4056-6/10 $26.00 © 2010 IEEE
DOI 10.1109/FCCM.2010.18

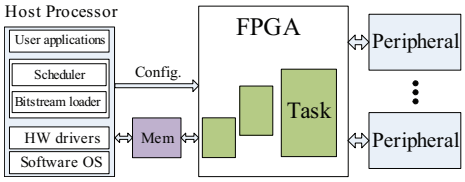

Figure 1. The system model

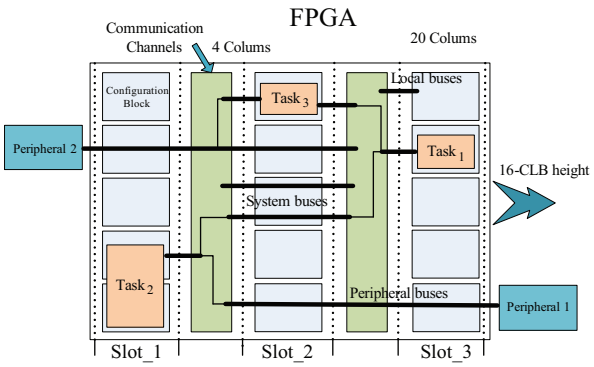

Figure 2. The 2D area with comm. structure

unit in our algorithm. Each task is only allowed to be allocated in the neighboring configuration block(s) in the same slot.

The communication structure consists of communication channels, local buses, system buses and peripheral buses. Each configuration block has its own local buses which connect to the adjacent communication channel(s). Between two adjacent slots, we add a communication channel where all the buses can be connected. The system buses horizontally expand the FPGA and connect to all communication channels, which allow communications between tasks in non-adjacent slots. The peripheral buses are the same as system buses but with an end connecting to peripherals. An example of communication among *Task* 1, 2, 3 and two external peripherals is shown in Figure 2: $Task_1$ and $Task_2$ are connected to the same system bus for data exchange; $Task_1$ and $Task_3$ are directly connected via the local buses in the channel; $Task_2$ and $Task_3$ are connected to the two peripheral buses respectively for communicating with external devices.

With this communication infrastructure, a communication route for a scheduled task is built by connecting the local buses to the destination buses. Since all the bus connections must be built within the communication channels and the locations of all the buses are known when the area partitioning is decided, the necessary communication connections in the channels can be derived regardless of the task configuration and allocation. The communication protocols among tasks could be designed according to various tasks' specifications and the relatively fixed communication structure.

*3) **Hardware task model**:* We combine task size, execution time, configuration time and communication time into the task model in order to reflect both spatial and temporal requirements. Each task supposes to occupy a rectangular area (Width x Height) of configurable logic blocks (**CLB**s). It arrives at arbitrary time t_a and requires its configuration time (t_{config}), execution time (t_{exe}) on the FPGA and communication time (t_{comm}). Since each task is only allowed to be allocated in a slot, tasks will have the same width and the size of the tasks are represented only by their heights.

The task model has two variants as shown in Figure 3. The horizontal direction reflects the spatial requirement which is the task's size represented by its height. The vertical direction stands

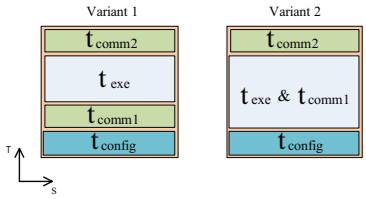

Figure 3. The task model

for the temporal requirement. The t_{comm1} stands for the loading of raw data and the t_{comm2} is the time to send the processed data to an expected destination. The variant 2 which mixes the t_{comm1} and t_{exe} together represents tasks who have intensive memory access during their executions.

B. Problem analysis and solutions

*1) **Spatial and temporal requirements for configurable resources**:* When our 2D area and task model are applied, the utilization of the FPGA can be mapped into the strip packing model described in [2]. A task's spatial and temporal requirements for configurable resources can then be met by assigning it a proper rectangular allocation in the strip model.

In Figure 4(a), the strip model reflects the utilization of the 3 slots shown in Figure 2 (the utilization of a slot is referred to as **Uslot**). The vertical axis represents time and the horizontal axis stands for the height of the slot. Since a configuration block is treated as the minimum configurable unit, the height of all slot is from '1' to '5'. All *Uslot*s are mapped in the strip packing model by adjusting their horizontal axis ranges (e.g. the *Uslot* 2 is mapped from '6' to '10').

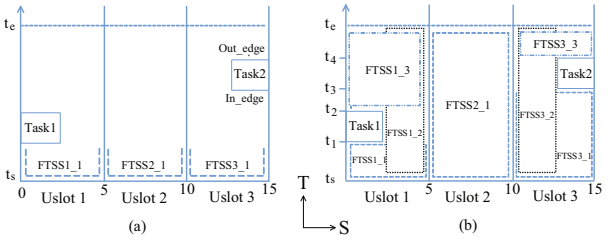

Figure 4. mFS for the 2D area scheduling

Because the strip has temporal and spatial coordinates, we define the maximum free rectangle in the strip as free time and space slot (**FTSS**). The previously proposed flow scanning (**FS**) algorithm [7] is modified to find all available *FTSS*s for arriving tasks in the strip. The modified *FS* (**mFS**) sets the searching interval for each arriving task, which is the time period between its arriving time (t_s) and the expected end time (t_e) as shown in Figure 4(a). By setting initial $FTSS$ in each $Uslot$, the mFS will find all $FTSS$s as shown in Figure 4(b). The detail mechanism of the mFS has been described in [8][5].

*2) **Temporal requirements for the allocation-confirmed resource**:* Certain resources such as configuration port, system buses and peripheral buses are shared among tasks. An efficient schedule has to take this sharing into account. We give an example shown in Figure 5.

Assume there is one configuration port and one peripheral bus on the FPGA. Without taking these constraints into account, two data independent tasks $Task_1$ and $Task_2$ will be scheduled starting at t_s as shown in Figure 5(a). However it is obvious that such scheduling results in an access conflict when the two tasks try to access the configuration port(t_{config}) and the peripheral bus(t_{comm1}) at the same time. A realistic scheduling result is shown in Figure 5(b) and

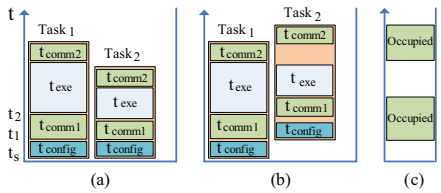

Figure 5. Communication buses constraint

Figure 5(c) shows the occupation status of the peripheral bus. In our scheduling algorithm, the availabilities of each allocation-confirmed resource is maintained in a linked list by the software OS and the scheduler together. The configuration port constraint was studied in [8] and we now extend it to include communication requirements and task dependency.

IV. The CASA scheduling algorithm

The proposed Communication Aware online task Scheduling Algorithm (**CASA**) has three basic steps. When scheduling an arriving task, first, the *mFS* algorithm is called to find all available $FTSS$s in the strip model between a search interval. Then these $FTSS$s are checked and adjusted if any conflict with the configuration port is found. Finally, the task is scheduled into a $FTSS$ according to its communication requirements.

In this paper, since we focus on the communication requirements when tasks running on the FPGA, we assume that all hardware tasks will be scheduled on the FPGA. This can be accomplished by setting each task's expected end time $t_e = t_{latest} + t_{config} + t_{comm1} + t_{exe} + t_{comm2}$. The t_{latest} represents the latest completion time of all already scheduled tasks. The other terms have been defined earlier in this paper.

In *CASA*, task arrivals are represented by task graphs. Each task graph represents a part of an application as well as any data dependency between tasks. Tasks in each task graph are supposed to process some data from an external peripheral. We assume that each task has at most one child task and one ancestor task (e.g. $Task$ 1, 2 and 3 in Figure 6(a)).

The scheduling heuristic used in $CASA$ is called "locked communication scheduling" (**LCS**). If a task is successfully scheduled, all the needed communication buses have to be assigned to the task during the required periods, in other words, the required buses are "locked" for the task during scheduled times. *LCS* schedules $Task_i$ in the first-found suitable $FTSS$ and assigns t^i_{config} ($\mathbf{t^i_{abc}}$ is used to represent $\mathbf{t_{abc}}$ of $\mathbf{Task_i}$) at the bottom of the $FTSS$ where the availability of the configuration port is checked in the second step. Then we will explain how *LCS* handles the communication requirements of each task in a task graph by giving an example shown in Figure 6. In order not to overload the figure, the configuration time is not shown.

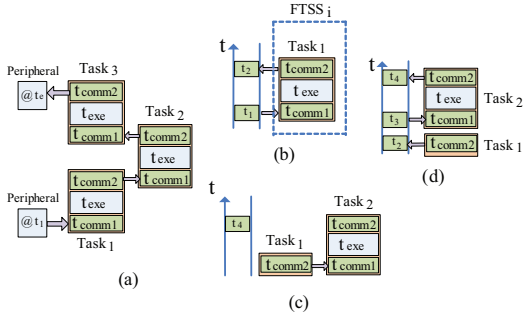

Figure 6. An example of LCS

The first task in a task graph is assumed to download raw data from an external device as shown in Figure 6(a). In an available $FTSS_i$ where $Task_1$ can be assigned to, *LCS* checks whether the required peripheral bus is available and determines the times for t^1_{comm1} and t^1_{comm2}. If t^1_{comm2} completes earlier than $FTSS_i$, $Task_1$ is successfully scheduled as shown in Figure 6(b). $Task_1$ is supposed to send intermediate data to $Task_2$ during t^1_{comm2}. Due to the unknown scheduling result of $Task_2$ at this moment, the possibility of building a direct connection between the two tasks is uncertain. Therefore, when scheduling $Task_1$, the peripheral bus is reserved for $Task_1$ at t^1_{comm2} in order to allow it to temporally store the intermediate data in the peripheral buffer. This is applied to any task ($Task_i$) in a task graph.

For the second task, *LCS* needs to handle the data dependency between tasks. For any two data dependent tasks (represented by $Task_1$ and $Task_2$), there are 3 possible situations in *CASA* as shown in Figure 7. *LCS* first tries to schedule $Task_2$ in "situation A" where the two tasks are allocated in adjacent slots and t^1_{comm2} and t^2_{comm1} are scheduled into the same time period. $Task_1$ can therefore send the intermediate data directly to $Task_2$ via the local buses. If "situation A" can not be met, *LCS* then tries to schedule $Task_2$ in "situation B" where the two tasks are allocated in non-adjacent slots and the system bus needs to be available at t^2_{comm1}. Hence, the two tasks will communicate directly via the system bus they are connected to. In any of the above two situations, the peripheral bus reserved at t^1_{comm2} for $Task_1$ can be released. The scheduling result of these two situations are shown in Figure 6(c). If "situation B" can not be applied either, *LCS* will schedule $Task_2$ in "situation C" where $Task_1$ stores the intermediate data in a peripheral buffer and $Task_2$ will later read these data. Figure 6(d) shows the scheduling of "situation C". In order to simplify the question, we assume the buffers are big enough to temporally store the intermediate data. It can be observed that from "situation A" to 'C', more time and system resources (e.g. peripheral buffer) are consumed.

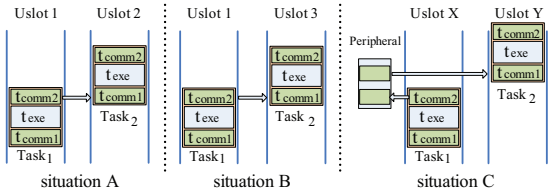

Figure 7. Situations for 2 dependent tasks

The scheduling processes of the rest tasks in a task graph (e.g. $Task_3$) are similar to that of $Task_2$. In *LCS*, if a task is modeled by variant 2 of the task model, *LCS* will schedule it only in the "situation C" because of its intensive access to a peripheral.

V. Experimental evaluation

To our best knowledge, no previously proposed online task scheduling algorithm targeting 2D FPGA area explicitly treated communication as a specific constraint. Therefore, we compare *CASA* with the recently proposed **CR** [6] which showed the best performance among the scheduling algorithms not only targeting 2D area but also assigning tasks with exact allocation and execution time on the FPGA. Since CR does not take into account the communication and configuration constraints, in the simulation, we added an idealized *CASA* (*CASA*-ideal) which also omits these constraints.

We will use the application completion time (**ACT**) and single task waiting time (**STWT**) to evaluate the different scheduling algorithms. The execution times of all algorithms will be given as well. The $STWT$ is defined as the time interval from a task's arriving time to its starting time.

The target FPGA model is Xilinx Virtex4 XC4VF100 FPGA which contains 160(rows) x 68(columns) CLBs. The 2D area and communication infrastructure on the target FPGA are similar as the ones shown in Figure 2: each slot occupies 20 columns of CLBs and each communication channel uses 4 columns of CLBs. Each configuration block covers the height of 16 CLBs and its configuration time is about 0.34ms. All the buses in the communication infrastructure can be implemented by using Xilinx slice-based busmacros [9].

Thanks to DWARV hardware compiler [10] and "Molen" hardware platform [11], the necessary information (e.g. task size and execution time) on the real hardware tasks (e.g. DCT and AES) was collected to generate sufficiently large synthetic workloads.

Task sizes are uniformly distributed in $[320..1920]$CLBs. The execution times are distributed in $[1.00..50.00]$ms and the t_{comm1} and t_{comm2} of each task are distributed in $[5\%..15\%]$ of its execution time. The number of tasks in each task graph is in the range of $[1..5]$ and the time interval for the arriving task graph is distributed in $[1..5]$ms. There are around 30 task graphs in each application which requires $[1..3]$ peripheral(s). Peripherals's interfaces are randomly chosen in $[32, 64, 96, 128]$ bits and there is a 128-bit system bus.

Currently, the proposed algorithm is running in simulation. All evaluated algorithms were programmed using C, and executed under Windows Vista with Intel(R) Core(TM)2 Duo CPU @ 2.10GHz.

A. Comparison of ACT and $STWT$

In the simulation, in total 10000 applications were processed by each scheduling algorithm. The average ACT and STWT of each algorithm is shown in Figure 8 and Table I gives their standard deviations.

CASA-config only considers the configuration time constraint [8]. "V_1" means the hardware tasks are only modeled with variant 1 of the task model and "V_{1+2}" is for the tasks modeled with both variant 1 and 2 of the task model. In our simulation, on the average 26.7% of the tasks in an application are modeled with variant 2.

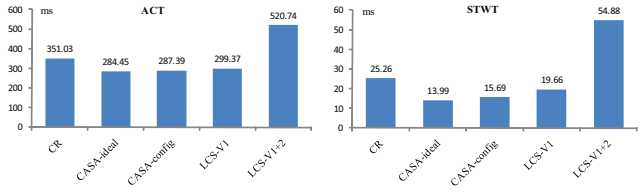

Figure 8. Average ACT and STWT

As shown in Figure 8, when only variant 1 of the task model is used, the *CASA* based algorithms outperform the CR. This is mainly because CR bring more area fragmentation. The *CASA*-config and LCS-V_1 show a small increasing ACT and STWT compared to *CASA*-ideal, this is due to the sequential access to the allocation-confirmed resources which, to some extent, serializes the applications' executions. This can be more clearly observed when variant 2 of the task model is applied (LCS-V_{1+2}).

For all ACT and STWT results of the 10000 applications, only these for *LCS*-V_{1+2} are distributed in relatively large ranges as shown in Table I. This is because they are highly effected by the random factors such as number of tasks modeled with variant 2 of the task model and the appearance frequency of these tasks.

Table I. Std. dev. of average ACT, STWT

	CR	CASA-ideal	CASA-config	LCS-V_1	LCS-V_{1+2}
ACT	42.05	33.69	33.59	34.88	120.23
STWT	6.55	3.73	3.80	4.39	21.85

B. Execution time of algorithms

The execution time of each algorithm is defined as the time used by an algorithm to schedule a task. The average execution times are 40μs for $CASA$-$ideal$, 44μs for $CASA$-$config$, and 37μs for both LCS-V_1 and LCS-V_{1+2}. The *CASA* based algorithms is about 20x faster than CR (847μs).

VI. Conclusion and future work

In this paper, we highlighted *CASA*'s ability to handle the communication constraints when scheduling tasks. In the future, our work will focus on: (i) implementing *CASA* into our FPGA-based configurable platform; (ii) investigating in the online task scheduling taking correlation analysis of multiple allocation-confirmed resources.

Acknowledgment: This work is sponsored by the hArtes project (IST-035143) supported by the Sixth Framework Programme of the European Community under the thematic area "Embedded Systems".

References

[1] Y. Lu, T. Marconi, G. Gaydadjiev, and K. Bertels, "A Self-adaptive on-line Task Placement Algorithm for Partially Reconfigurable Systems," in *Proceedings of the 22nd Annual International Parallel and Distributed Processing Symposium (IPDPS 2008)*, Miami, Florida, USA, April, 2008, pp. 8.

[2] C. Steiger, H. Walder, and M. Platzner, "Operating systems for reconfigurable embedded platforms: Onlline scheduling of real-time tasks," in *IEEE Transactions on Computers*, vol. 53, Nov. 2004, pp. 1393–1407.

[3] S. Banerjee, E. Bozorgzadeh, and N. Dutt, "Exploiting application data-parallelism on dynamically reconfigurable architectures: Placement and architectural considerations," in *IEEE Transactions on Very Large Scale Intergration Systems*, vol. 17, 2009, pp. 234–247.

[4] W. Fu and K. Compton, "Scheduling intervals for reconfigurable computing," in *Proceedings of the 16th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2008)*, Stanford University, Palo Alto, California, USA, April, 2008, pp. 87–96.

[5] Y. Lu, T. Marconi, K. Bertels, and G. Gaydadjiev, "Online task scheduling for the FPGA-based partially reconfigurable systems," in *Proceedings of the the 5th International Workshop on Applied Reconfigurable Computing (ARC 2009)*, Karlsruhe, Germany, March, 2009, pp. 216–230.

[6] X. Zhou, Y. Wang, X. Huang, and C. Peng, "Fast on-line task placement and scheduling on reconfigurable devices," in *International Conference on Field Programmable Logic and Applications (FPL 2007)*, Amsterdam, the Netherlands, August, 2007, pp. 132–138.

[7] Y. Lu, T. Marconi, G. Gaydadjiev, and K. Bertels, "An efficient algorithm for free resource management on the fpga," in *Proceedings of Design, Automation and Test in Europe (DATE 08)*, Munich, Germany, March 2008, pp. 1095–1098.

[8] Y. Lu, T. Marconi, K. Bertels, and G. Gaydadjiev, "Technology aware online task scheduling algorithms for the fpga-based partially reconfigurable systems," in *Submitted to ACM Transactions on Reconfigurable Technology and Systems*.

[9] "Xilinx, partial reconfiguration early access software tools," in *http://www.xilinx.com/support/prealounge/protected/index.htm*.

[10] Y. Yankova, G. Kuzmanov, K. Bertels, G. Gaydadjiev, Y. Lu, and S. Vassiliadis, "DWARV: Delft workbech automated reconfigurable VHDL generator," in *International Conference on Field Programmable Logic and Applications (FPL)*, August, 2007, pp. 697–701.

[11] S. Vassiliadis, S. Wong, G. Gaydadjiev, K. Bertels, G. Kuzmanov, and E. M. Panainte, "The MOLEN polymorphic processor," in *IEEE Transactions on Computers archive*, vol. 53, Nov. 2004, pp. 1363–1375.