Scheduling Dynamic Hard Real-Time Task Sets on Fully and Partially Reconfigurable Platforms

Sangeet Saha, Arnab Sarkar, *Member, IEEE*, and Amlan Chakrabarti, *Senior Member, IEEE*

***Abstract*—Reconfigurable systems are increasingly being employed in a large class of today's heterogeneous real-time embedded systems which often demand satisfaction of stringent timeliness constraints. However, executing a set of hard real-time applications on reconfigurable systems such that all timing constraints are satisfied while also allowing efficient resource utilization requires effective scheduling, mapping and admission control strategies. This letter presents methodologies for scheduling periodic hard real-time dynamic task sets on fully and partially reconfigurable field-programmable gate arrays (FPGAs). The floor of the FPGA is assumed to be statically equipartitioned into a set of homogeneous tiles (each of which act as individual processing elements or PEs) such that any arbitrary task of the given task set may be feasibly mapped into the area of a given tile. Experimental results reveal that the proposed algorithms are able to achieve high resource utilization with low task rejection rates over a variety of simulation scenarios.**

***Index Terms*—Field-programmable gate arrays (FPGAs), full reconfiguration, partial reconfiguration, proportional fair scheduling, real-time task scheduling.**

I. Introduction

RECONFIGURABLE systems attempt to leverage and combine the benefits of flexibility of general purpose processors along with the performance efficiency of a dedicated hardware [1]. Today, reconfigurable systems are being used in a plethora of embedded devices ranging from avionic and automotive applications [2] to synthetic vision, object tracking, etc. [3]. Many of these applications are often safety-critical and may be characterized by varying degrees of timeliness constraints.

As an interesting example, reconfigurable field-programmable gate arrays (FPGAs) may be used as a performance efficient back up platform for real-time tasks in a complex safety-critical system. Suppose, this backup platform is activated whenever one or more processors in the system fail and should assume the responsibility of executing the real-time tasks that were previously running on the failed processors. Therefore, it is essential to have well defined scheduling methodologies, feasibility criteria and admission control mechanisms for real-time task sets in these reconfigurable platforms such that all timing constraints may be met while taking care of the reconfiguration overheads and also allowing efficient resource utilization.

The problem of efficient task scheduling and resource management in FPGAs has attracted considerable interest among the research community in the last few years [4], [5]. A majority of the proposed algorithms focus on the problem of efficient placement of tasks on the hardware floor such that fragmentation and inter-task communication overheads are minimized. However, these works are mostly limited to the handling of soft real-time tasks. A hardware task relocation scheme has been discussed in [6], where a running task is allowed to be preempted within a hardware module and later may be relocated in another module depending upon its required remaining execution area. In [7], authors presented a heuristic for runtime task scheduling by managing multitasking in reconfigurable computers. Another scheduling approach, *Reuse and Partial Reuse* was proposed in [8] where the authors investigated the scheduling impact of reusing already placed tasks. Danne and Platzner [9] consider the problem of scheduling periodic real-time tasks on FPGAs and proposed EDF-Next Fit (EDF-NF), a variant of the EDF [10] algorithm, which uses the concept of master processes (servers) to reserve area and execution time for tasks. However, as the paper reveals, being based on EDF, this algorithm cannot achieve a resource utilization of more than 50% in case of generic systems. All these techniques and other methods as in [11]–[13] only consider partially reconfigurable platforms. In comparison, there is very little literature that addresses the scheduling problem in fully reconfigurable systems.

Recently, a few interesting periodic real-time scheduling strategies like ERfair [14], DP-Fair [15] etc. that allow optimal resource utilization have been proposed for general purpose multiprocessor systems. Given a set of n tasks $\{T_1, T_2, \ldots, T_n\}$, with each task T_i having a computation requirement of e_i time units, required to be completed within a period of p_i time units from the start of the task, the ERfair algorithm defines the $weight$ of T_i as $wt_i = \frac{e_i}{p_i}$ and prescribes that task allocations should be managed in such a way that each task is executed at a consistent rate proportional to its own weight. More formally, let the start time of a task T_i be s_i. Then ERfairness guarantees the following for every task T_i: *at the end of any given scheduling quanta or time slot* t, $s_i \leq t \leq s_i + p_i$, *at least* $\frac{e_i}{p_i} * (t - s_i)$ *of the total execution requirement of* e_i *must be completed*. Obviously, for such a criterion to be guaranteed in a system of m processors, we must have $\sum_{i=1}^{n} e_i/p_i \leq m$. One of the major drawbacks of ERfair

Manuscript received December 13, 2014; accepted January 14, 2015. Date of publication January 23, 2015; date of current version February 24, 2015. This work was supported in part by the TCS Research Fellowship Award, granted to S. Saha. This manuscript was recommended for publication by W. Zhao.

S. Saha and A. Chakrabarti are with the A. K. Choudhury School of IT, University of Calcutta, Kolkata, India (e-mail: sasakc_s@caluniv.ac.in; amlanc@ieee.org).

A. Sarkar is with the Department of Computer Science & Engineering, IIT Guwahati, Guwahati 781039, India (e-mail:arnabsarkar@iitg.ernet.in).

Digital Object Identifier 10.1109/LES.2015.2396069

is that they suffer unrestricted migration and context-switch related overheads which turn out to be reasonably expensive in most systems.

DP-Fair [15] attempts to lower the number of context switches while concurrently preserving scheduling optimality by enforcing ERfairness constraints to be satisfied only at task period / deadline boundaries. The approach partitions time into slices, demarcated by the arrivals and departures of all the jobs in the system. Within a time slice, each task is allocated a workload equal to its proportional fair share. Jobs within a slice are scheduled using the least laxity first (LLF) [10] algorithm. DP-Fair when implemented along with DP-Wrap [15] guarantees atmost $m - 1$ context-switches within any given time slice.

However, it is not possible to devise periodic real-time scheduling algorithms for reconfigurable systems by directly employing strategies like DP-Fair primarily due to their inherent architectural constraints and reconfiguration overheads. This work assumes a 2-dimensional reconfigurable resource area in which the device floor is equi-partitioned into a set of m homogeneous tiles (each of which act as individual processing elements) such that any arbitrary task may be feasibly mapped into the area of a given tile. We consider both fully reconfigurable systems (with worst-case reconfiguration time for a given system being t_{frg}; typically, t_{frg} ranges between a few milli secs. to tens of milli secs.) in which all the tiles must be simultaneously reconfigured as well as run-time partially reconfigurable systems (with worst-case reconfiguration time t_{prg}; typically, t_{prg} is a few hundred micro secs.) which allow one or more tiles to be reconfigured while the hardware designs in the other parts run uninterrupted.

It may be noted that any real-time scheduling strategy for these reconfigurable systems will incur a reconfiguration overhead (t_{frg} and t_{prg} for fully and partially reconfigurable systems respectively) for every context switch suffered and hence the techniques that are designed must appropriately incorporate these overheads. This letter presents novel periodic real-time scheduling algorithms deadline partitioning scheduler for fully reconfigurable systems (*DPSFR*) and deadline partitioning scheduler for partially reconfigurable systems (*DPSPR*) for fully and partially reconfigurable target platforms that attempt to maximize achieved resource utilization. Experimental results reveal that both DPSPR and DPSFR are able to achieve high resource utilization with low task rejection rates over a variety of simulation scenarios.

II. Scheduling Strategy

Given a set of n tasks $\{T_1, T_2, \ldots, T_n\}$ to be scheduled on m homogeneous partitions (or tiles) $\{V_1, V_2, \ldots, V_m\}$ of a reconfigurable system, both DPSFR and DPSPR maintain the time partitioned into slices (the rth time slice is denoted by ts_r) demarcated by all the deadlines of all tasks in the system. At any time slice boundary, each task T_i is allocated a share shr_i (in proportion to its weight w_i) for the next time slice ts_r as follows: $shr_i = \frac{e_i}{p_i} * ts_r$. Given the sum of the shares of all tasks sum_shr ($= \sum_{i=1}^{k} shr_i$, where k denotes the number of active tasks at ts_r having share values at least 1 ($shr \geq 1$)) and the total system capacity ($ts_r * m$) over the interval ts_r, the following necessary feasibility condition must be satisfied for scheduling to proceed:

$$sum\_shr \leq ts_r * m. \tag{1}$$

Now, the actual schedule and the sufficiency conditions vary for DPSFR and DPSPR.

A. The DPSFR Working Principle

It may be noted that in case of fully reconfigurable systems, task context switches are only allowed to occur synchronously in all the available partitions and each such system wide context switch event will require a full reconfiguration. The total scheduling overhead that must be incurred by the overall system due to this context switch event is given by: $t_{frg} * m$, where t_{frg} denotes the full reconfiguration time. Now, this overhead may only be possibly compensated by the available system wide slack (if any) at time slice ts_r and is given by: $ts_r * m - sum\_shr$. This brings us to a weaker necessary feasibility condition for fully reconfigurable systems (than the one presented in equation (1)

$$t_{frg} * m \leq ts_r * m - sum\_shr. \tag{2}$$

The total number of context switches C_{frg} that the system can afford during the interval ts_r is

$$C_{frg} = \lfloor \frac{ts_r * m - sum\_shr}{t_{frg} * m} \rfloor. \tag{3}$$

Therefore, our *Deadline Partitioning Scheduler for Fully Reconfigurable Systems (DPSFR)* first checks the condition in equation (2) and then calculates the number of full reconfigurations that are possible (C_{frg}) using equation (3). One of these reconfigurations will of course be required to initially allocate tasks to the m partitions at the beginning of time slice ts_r. The rest of the reconfigurations are used to allow $C_{frg} - 1$ equally spaced context switches within the interval ts_r. The time period between any two such consecutive context switch events is called a *frame* and is denoted by G_{frg}. It may be noted that a single task must be allocated a dedicated partition or tile at least for the duration of a single frame. At each frame boundary within the interval ts_r, DPSFR selects m tasks with the highest remaining shares and assigns them for execution on the m available tiles for the duration of the frame. The number of frames required by a task T_i to complete execution of its share shr_i is: $\lceil \frac{shr_i}{G_{frg}} \rceil$. The sufficient condition for feasible scheduling within a time slice may thus be formulated as

$$\sum_{i=1}^{k} \lceil \frac{shr_i}{G_{frg}} \rceil \leq C_{frg} * m. \tag{4}$$

When a new task T_i arrives, the above condition is checked for all future time slices until the total execution requirement of the task is fulfilled. T_i is rejected if equation (4) fails in any future time slice. We now present an example to illustrate the DPSFR scheduling mechanism before discussing the DPSPR technique for runtime partially reconfigurable systems.

Example 1: Let us consider 6 tasks $T_1, T_2, \ldots, T_6$, having weights ($\frac{e_i}{p_i}$) 24/60, 36/90, 24/60, 72/90, 72/90 and 36/90. Also, we have $m = 4$ (4 tiles), $t_{frg} = 6$ time slots (time slot size =

1 ms) and the time slice $ts_0 = 60$. As we have 4 tiles which act as concurrent processing elements, the system workload becomes: $\frac{1}{4} * \sum_{i=1}^{6} \frac{e_i}{p_i} * 100\% = 80\%$. The task shares to be executed within the interval ts_r are: $shr_1 = shr_2 = shr_3 = shr_6 = 24$ and $shr_4 = shr_5 = 48$. As $sum\_shr = 192$, $ts_0 * m = 240$ and $t_{frg} * m = 24$, both the necessary conditions in equations (1) and (2) are satisfied. The total number of context switches possible is, $C_{frg} = \lfloor (240 - 192)/24 \rfloor = 2$ [refer equation (3)]. Hence, we may allow two frames in the time slice, each of length $G_{frg} = (60 - 12)/2 = 24$. It may be noted that in this case, the sufficient condition in equation (4) is also satisfied. After a full reconfiguration in the interval 0 to 6 time slots, the 1st frame executes with the tasks T_4, T_5, T_1, T_2 within 6 to 30 time slots. The 2nd full reconfiguration occurs between 30 and 36 time slots. Finally, tasks T_3, T_4, T_5, T_6 are selected and executed in the 2nd frame within the time slot interval 36 to 60. All tasks successfully complete their shares within the time slice. It may be observed however that if the share (shr_i) of any task be increased by just 1, the sum of shares become $sum\_shr = 193$ and only one frame may be accommodated within the time slice [C_{frg} becomes 1; refer equation (3)]. So, the sufficient condition in equation (4) fails, making scheduling infeasible.

B. The DPSPR Working Principle

In terms of real-time scheduling and resource utilization, the principal architectural advantage of a run-time partially reconfigurable system is that, unlike full reconfigurable systems, a context switch (reconfiguration) do not require to be a global event and may be localized to individual partitions. In addition, the partial reconfiguration overhead t_{prg} is generally much lower than a full reconfiguration overhead t_{frg}.

After checking the necessary condition in equation (1), DPSPR partitions the task set into m disjoint subsets using a strategy similar to DP-Wrap [15] as follows: the tasks are allocated (one tile at a time) in any order starting from the first tile V_1 such that the combined sum of task shares along with the reconfiguration overheads in each such tile is less than the time slice interval ts_r. Like DPSFR, all tiles incur a reconfiguration overhead t_{prg} at the start of the time slice; hence, the initial remaining capacity rc_i of each tile V_i is given by: $rc_i = ts_r - t_{prg}$. After a task say T_j is allocated to a tile V_i, its remaining capacity becomes: $rc_i = rc_i - shr_j - t_{prg}$. Such an allocation is however allowed only if $rc_i - shr_j \geq 0$. Otherwise, T_j must be partitioned into two parts; the first part T_{j1} with a share value $shr_{j1} = rc_i$, is executed in V_i while the second part with a share $shr_{j2} = shr_j - rc_i$, is executed in the next tile V_{i+1} provided $i < m$. If V_i is the last available tile ($i = m$), then system capacity is exhausted and scheduling cannot proceed in time slice ts_r. Residual remaining capacity (if any) after completion of the task allocation process may possibly be used to accommodate dynamically arrived tasks in the middle of the time slice.

Example 2: Considering a run-time partially reconfigurable scenario, let us again take the same 4 tile platform and the same set of 6 tasks as used in example 1, with the exception that task T_4's execution requirement e_4 is incremented by 2 from 72 to 74 making its share value shr_4 within the first time slice ($ts_0 = 60$ ms) increase by 1 from 48 to 49. The sum of shares (sum_shr) now become 193. As discussed in example 1, this slightly modified task set can't be scheduled using DPSFR on fully reconfigurable systems. Assuming a partial reconfiguration overhead $t_{prg} = 1$ ms, the initial remaining capacity of each of the four partitions will be $ts_0 - t_{prg} = 59$ ms. Fig. 1 shows the schedule generated by DPSPR. It may be noted that while T_1, T_2 and T_5 gets allocated fully on a single tile, T_3, T_4 and T_6 partially executes on two different tiles.

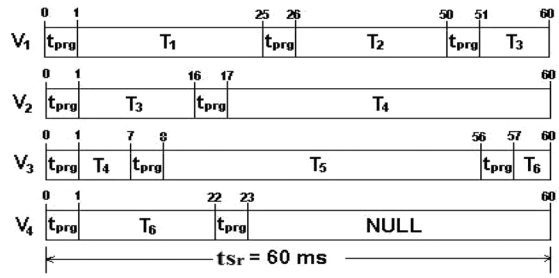

Fig. 1. Task Scheduling under DPSPR.

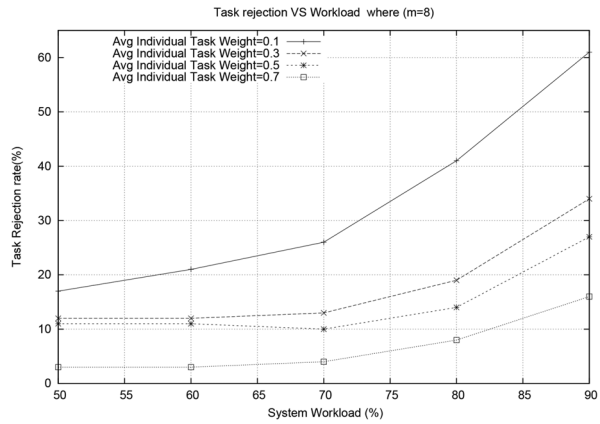

Fig. 2. TRR versus WL; $m = 8$: DPSFR.

III. Experiments and Results

The performance of the DPSFR and DPSPR have been evaluated by conducting simulation based experiments using randomly generated periodic task sets whose weights ($\frac{e_i}{p_i}$) and execution periods (p_i) have been taken from normal distributions. The principal performance metric used is *Task Rejection Rate (TRR)* which is defined as the percentage of the total number of tasks rejected by the system over the entire schedule length. That is

$$TRR = (\nu/N) * 100 \tag{5}$$

where, ν and N denote the total number of tasks rejected and arrived, respectively. Data sets for various values of average individual task weights and average total system loads have been generated on systems containing 2 to 10 tiles. Each result is generated by running 100 different instances of each dataset type and then taking the average over these 100 runs. The total schedule length is 100 000 time slots in all our experiments.

Figs. 2 and 3 depict plots showing *task rejection rates* (TRR) suffered by DPSFR on fully reconfigurable systems (having

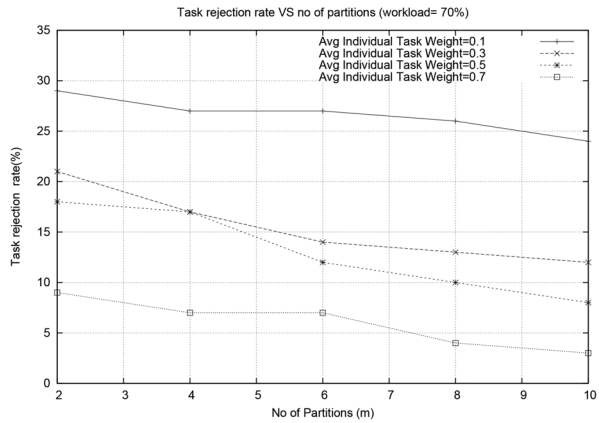

Fig. 3. TRR versus m; $WL = 70\%$: DPSFR.

TABLE I
: TRR VS. t_{frg}: DPSFR; AVERAGE TASK WEIGHT $\omega_i = 0.3$, NUMBER OF TILES $m = 8$ AND SYSTEM WORKLOAD $WL = 70\%$

t_{frg} (ms)	6	12	18	24	30
TRR (%)	13	20	29	34	51

TABLE II
TASK REJECTION RATE (%): DPSPR

	m=2			m=4			m=8		
WL (%)	ω_i=0.1	ω_i=0.3	ω_i=0.5	ω_i=0.1	ω_i=0.3	ω_i=0.5	ω_i=0.1	ω_i=0.3	ω_i=0.5
50	0.02	0	0	0.01	0	0	0	0	0
60	0.36	0	0	0.06	0	0	0	0	0
70	2.78	0	0	1	0	0	0.18	0	0
80	6.56	1.57	1.05	4.83	0.17	0	3	0	0
90	14.50	2.26	5.15	11.6	0.95	0.65	9.75	0.17	0.14

ω_i: Avg. Individual Task Weight; m: Total Number of tiles; WL: Total System work load.

$t_{frg} = 6ms$) for task sets whose average individual weights (ω_i) vary between 0.1 and 0.7. While Fig. 2 shows the variation of TRR with increase in total workload (WL) from 50% to 90% on an eight tile system, Fig. 3 highlights the TRR variation as the number of tiles increase from 2 to 10 with total system load (WL) fixed at 70%. It may be observed from Fig. 2 that for a given value of ω_i, TRR increases with increase in WL. This is because as WL increases, the average number of tasks in the system also increases (as ω_i is same), resulting in the LHS ($\sum_{i=1}^{k} \lceil \frac{shr_i}{G_{frg}} \rceil$) of equation (4) to become higher. Due to this, the probability of failure of the sufficiency condition [equation (4)] increases. From Fig. 3, it may be noted that for a given value of ω_i, TRR gradually decreases with increase in the number of tiles m. This may be attributed to the fact that for a particular WL, although the approximate number of rejected tasks ν remains same, the total number of tasks N increases with higher values of m, thus lowering the value of TRR [refer equation (5)].

Table I shows the effect of the variation of the full reconfiguration overhead t_{frg} on TRR. As is obvious from equation (3), the value of C_{frg} decreases as t_{frg} increases, causing a higher number of tasks to be rejected.

Table II summarizes the results for TRR on runtime partially reconfigurable systems. It may be observed that due to much lower reconfiguration overheads t_{prg} and the ability to asynchronously reconfigure individual tiles, DPSPR is able to achieve much lower TRR as compared to DPSFR. Also, due to similar reasons as for fully reconfigurable systems, the TRR values increase with increasing system loads and decrease with increasing number of tiles.

IV. CONCLUSION

In this letter, we presented methodologies for scheduling periodic hard real-time task sets on fully and runtime partially reconfigurable systems such that resource utilization is maximized. The scheduling feasibility conditions considering dynamic task sets have been discussed. We designed, implemented and evaluated the algorithms using simulation based experiments and the results are promising.

REFERENCES

[1] L. Shannon and P. Chow, "Leveraging reconfigurability in the hardware/software codesign process," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 4, no. 3, pp. 28:1–28:27, Aug. 2011.
[2] S. Sharma and W. Chen, Using model-based design to accelerate fpga development for automotive applications SAE Technical Paper, Tech. Rep., 2009.
[3] J. Jin, S. Lee, B. Jeon, T. T. Nguyen, and J. W. Jeon, "Real-time multiple object centroid tracking for gesture recognition based on fpga," in *Proc. 7th Int. Conf. Ubiquitous Inf. Manag. Commun.*, 2013, p. 80, ACM.
[4] Q.-H. Khuat and D. Chillet, "Communication cost reduction for hardware tasks placed on homogeneous reconfigurable resource," in *Proc. Design and Arch. Signal Image Process. Conf. (DASIP)*, Oct. 2013, pp. 265–270.
[5] A. Eiche, D. Chillet, S. Pillement, and O. Sentieys, "Task placement for dynamic and partial reconfigurable architecture," in *Proc. Design and Arch. Signal Image Process. Conf. (DASIP)*, Oct. 2010, pp. 228–234.
[6] A. Morales-Villanueva and A. Gordon-Ross, "Htr: On-chip hardware task relocation for partially reconfigurable fpgas," in *Proc. t 9th Int. Conf. Reconfigurable Comput.: Arch., Tools, Appl.*, 2013, pp. 185–196, ser. ARC'13.
[7] M. Fazlali, M. Sabeghi, A. Zakerolhosseini, and K. Bertels, "Efficient task scheduling for runtime reconfigurable systems," *J. Syst. Arch.*, vol. 56, no. 11, pp. 623–632, 2010.
[8] Y. Lu, T. Marconi, K. Bertels, and G. Gaydadjiev, "Online task scheduling for the fpga-based partially reconfigurable systems," in *Reconfigurable Computing: Architectures, Tools and Applications*. Berlin, Germany: Springer-Verlag, 2009, pp. 216–230.
[9] K. Danne and M. Platzner, "Periodic real-time scheduling for fpga computers," in *Proc. 3rd Int. Workshop Intell. Solutions Embed. Syst.*, May 2005, pp. 117–127.
[10] J. W. S. Liu, *Real-Time Systems*, 1st ed. ed. Upper Saddle River, NJ, USA: Prentice Hall, 2000.
[11] R. Pellizzoni and M. Caccamo, "Real-time management of hardware and software tasks for fpga-based embedded systems," *IEEE Trans. Comput.*, vol. 56, no. 12, pp. 1666–1680, Dec. 2007.
[12] M. Santambrogio, V. Rana, I. Beretta, and D. Sciuto, "Operating system runtime management of partially dynamically reconfigurable embedded systems," in *Proc. 8th IEEE Workshop Embed. Syst. Real-Time Multimedia (ESTIMedia)*, Oct. 2010, pp. 1–10.
[13] L. Bauer, A. Grudnitsky, M. Shafique, and J. Henkel, "Pats: A performance aware task scheduler for runtime reconfigurable processors," in *Proc. IEEE 20th Annu. Int. Symp. Field-Programmable Custom Comput. Mach. (FCCM)*, Apr. 2012, pp. 208–215.
[14] J. Anderson and A. Srinivasan, "Early-release fair scheduling," in *Proc. 12th Euromicro Conf. Real-Time Syst.Euromicro RTS 2000*, 2000, pp. 35–43.
[15] G. Levin, S. Funk, C. Sadowski, I. Pye, and S. Brandt, "Dp-fair: A simple model for understanding optimal multiprocessor scheduling," in *Proc. 22nd Euromicro Conf. Real-Time Syst. (ECRTS)*, Jul. 2010, pp. 3–13.