The CSI Journal on
Computer Science and Engineering
Vol. 11, No. 2 & 4 (b), 2013
Pages 15-23
Regular Paper

Performance Evaluation of Reusing Based Scheduling in On-line Reconfigurable Computing Systems

Hadi Shahriar Shahhoseini Maisam Mansub Bassiri Seyed Mehdi Mohtavipour

Electrical Engineering Department, Iran University of Science and Technology, Tehran, Iran

Abstract

Reconfiguration overhead is an important obstacle that limits the performance of on-line scheduling algorithms in reconfigurable computing systems and increases the overall execution time. Configuration reusing (task reusing) can decrease reconfiguration overhead considerably, particularly in periodic applications or the applications in which the probability of tasks recurrence is high. In this paper, we improve reusing technique in order to reduce reconfiguration overhead and decrease total execution time of the tasks. In reusing method, input tasks are divided into significant and non-significant tasks. Significant tasks are preserved on the RPU in order to be reused in near future. The main contributions in this paper include new task significance calculation and performance evaluation of resizing the RPU partitions. A large variety of experiments has been conducted on the proposed algorithm. Obtained results show when the recurrence of tasks is 40%, we have up to 14% improvement in overall execution time of resizable reusing based scheduling comparing reusing without resizing, and up to 25% improvement comparing scheduling without reusing.

Keywords: Reconfigurable Computing, On-Line Scheduling, Configuration Reusing, RPU Partitioning, Replacement Management, Probability of Recurrence.

1. Introduction

Reconfigurable computing (RC) is widely used in modern application due to developing the high speed reconfigurable device such as FPGAs that can be reconfigured at run-time and support partial reconfigurebility. RC systems include one CPU and one or several Reconfigurable Processing Unit (s) (RPU) [1, 2].

While RC systems can increase the computing performance, it also has many challenges ahead, rising in complex applications. A set of system services, forms a reconfigurable operating system (OS) [3, 4], try to overcome these challenges. Dynamic task allocation is an essential service that helps to efficiently perform applications.

In this paper resource management and specially task scheduling are studied as a part of a reconfigurable OS. In particular, we focus on the problem of on-line scheduling the tasks to the RPU in a RC system. We extend the reusing based placement algorithm, previously proposed by the authors of this paper in [5], to have more accurate significance criteria for task selection. Also the effect of partition resizing in the RPU is studied in detail to have more reduction in the reconfiguration overhead and improvement in the total execution time of the application. In the proposed method, input tasks are divided into significant and non-significant tasks. Significant tasks are preserved on the RPU in order to be reused in near future. The main contributions in this paper include proposing new task significance criteria, and evaluating resizable partitioning of RPU surface.

In fact, in this method, the significant tasks are place and preserved in RPU surface in such a way that fragmentation of RPU surface is reduced. Also, dynamic partition resizing and task replacement management are utilized to improve the performance of the proposed method and reduce the task rejection ratio in the application. As mentioned before, this technique may be beneficial for the application in which some tasks are repeated several times during the run time.

The rest of the paper is organized as follows. In section 2 the model for task and placement problem in RPU are described. In section 3 the previous and related works are reviewed. Section 4 presents reusing based scheduling algorithm and the new task grouping criteria. Simulation results are placed in Section 5 for evaluating the performance of task reusing and partition resizing. Finally the paper is concluded in section 6.

2. RC System Consideration

In this section firstly our task model is described; then some considerations regarding the task placement problem in RPU are briefly presented.

Generally a task is a function synthesized to digital circuit, and in RC system it can be programmed onto the reconfigurable device (RPU) or performed by CPU. A task has a size and a shape. The size gives the area requirement of the task in reconfigurable units. In this paper all task shapes are assumed to be rectangle and their execution times are known in advance. Each task is defined as a 6-tuple

$$T_i = (ec_i , er_i , w_i , h_i , a_i , d_i , r_i) \quad (1)$$

where ec_i , er_i , w_i , h_i , a_i , d_i and r_i denote execution time on CPU, execution time on RPU, width, height, arrival time, deadline and reconfiguration time of the task, respectively. Tasks can be real-time or non real-time. The deadline of a task is defined only for real-time tasks and determines the latest finish time of the task. Also, the tasks can be dependent or independent. Dependent tasks usually represented in a directed task graph. This paper focuses on real-time and independent tasks.

Regarding the execution of tasks, an important property is preemption; so task execution can be preemptive or non-preemptive. Although preemption techniques are useful, resulting in efficient and fast on-line scheduling algorithm, they seems to be too expensive for current reconfigurable technology, because preemption and resuming task, requires large context switches and needs additional external memory. Hence, this paper focuses on the non-preemptive task model i.e. once a task is loaded onto the device, it runs to completion.

Since the mapping of tasks to the RPU in RC systems, strongly depends on the area model of RPU, we should clarify the RPU area model. Generally there are four area models. The first one is flexible 2D area model which allows to allocate rectangular tasks anywhere on the 2D reconfigurable surface. This model yields to high device utilization and high flexibility while it has difficult scheduling and placement algorithms [6, 7]. The second model is partitioned 2D model where the reconfigurable surface is split into a statically-fixed number of allocation sites, so called blocks [8]. Each block can accommodate at most one task at a time. This model facilitates the scheduling and placement but causes internal fragmentation. In the third model, flexible 1D area model, tasks can be allocated anywhere along the horizontal device dimension and the vertical dimension is fixed. This model matches well current FPGA technology that is partially reconfigurable in vertical chip-spanning columns [9]. While the flexible 1D area model leads to low complexity placement problem, it suffers from both internal and external fragmentation. Finally the forth model is partitioned 1D area model. This model combines the characteristics of the second and third model. Again the disadvantage lies in the potentially high internal fragmentation. In this paper, we use the 1D area model and we assume tasks have known execution times and there is no dependency among them, no preemption is allowed, and 1D area model are considered for RPU.

On the other hand there are two different approaches are used for task placement: scanning method and management of free spaces. The scanning techniques are recognition-complete. This means they are optimal in the sense that they find a feasible placement for a new task if sufficient resources (free reconfigurable units) are available. Although high run-time complexity for large device area is a drawback of this technique, in 1D area model its run-time complexity is acceptable. In fact, in the worst case, W (the width of the RPU) potential placements have to be considered. In free space management techniques, for placement a task, only the free spaces are searched. So a list for free spaces is needed, that will be updated on each event. Some of space management techniques such as MER (maximal empty rectangle) management [6] are recognition-complete and some of them are not.

We consider a system that employs scanning-based placement for putting tasks in the RPU.

3. Related Work

There are two categories of scheduling in RC systems which affects on the algorithm of task placement in RPU.

The first one, which is simpler, is offline scheduling task placement and there is less time limitation for finding the optimal solution. However many researchers try to find the best solution in shorter time [10-12].

The second type of task scheduling is online scheduling. In online task scheduling, task arrives one after the other and the placer has not any task graph in advance. So the scheduler and placer should place the task in appropriate location in RPU or decide to reject it from RPU and send it to CPU in a short time. There are many researches address this type of scheduling in recent years [13-23]. Among all method of online scheduling, we focus on those that divide the task and/or the RPU to a number of partitions.

Ahmadinia et al. have proposed an integrated scheduling and placement algorithm in which the FPGA is divided into slots [16]. Task are placed in one of the slots based on their termination time. The width of the slots is also to be changed during runtime in order to improve the quality of placement.

Roman et al. have presented a technique in which FPGA area is divided into four partitions of different sizes [22]. Each partition has an associated queue where the hardware manager places each arriving task depending on its size, shape and real-time parameters as deadline. The algorithm may change the queue selection policy, partition strategies and the sizes or the number of partitions at run-time in order to adapt itself to the parameters of arriving tasks.

Hassanly et al. have proposed an analytical model for a partitioned RC system [23]. In their work partitions have been defined according to the size of tasks that would be placed on them.

In [18], a technique has been presented that reuse hardware tasks to reduce reconfiguration overhead. The

proposed algorithm migrate the tasks between hardware and software resources and gives the priority to the hardware tasks.

Reusing that is used in some previous researches, is a technique that tries to reduce the configuration overhead when a synthetic task may repeated in near future [5, 18, 24].

4. On-line Scheduling and Placement Algorithm

Most of proposed algorithms for scheduling and placement such as [6, 18, 26, 27] maintain the list of free spaces on RPU surface and update the list at any task arrival and task termination time. In such methods, once the execution of a task finishes, the task is removed from RPU surface and its area is added to free space list. Therefore, those algorithms can not exploit configuration reusing (task reusing) as much as possible. figure1a shows an example in which task T'_2 is an instance of T_2 (a copy of T_2) that arrives at time a'_2. In this situation this task should be reconfigured again on the free area of RPU. In this figure, dotted part of the tasks (R_i) indicate the reconfiguration time of the tasks. Reconfiguration overhead (Reconfiguration time) is an important obstacle that limits the performance of the RC systems and increases the overall execution time of the tasks. Reconfiguration overhead can considerably be decreased configuration reusing, particularly in applications in which some tasks are repeated frame by frame or packet by packet such as audio and video processing, cryptography, network applications and so on. Fine granularity increases the probability of task repetition in applications.

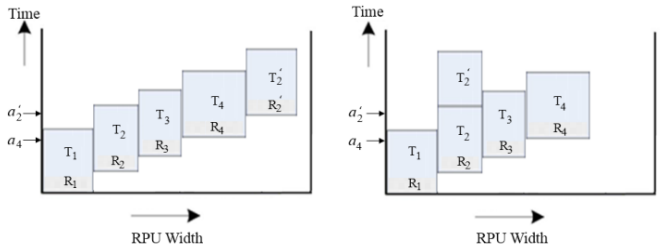
In this paper we try to improve on-line scheduling, by reusing a group of tasks specified by new task selection criteria. In reusing technique, some of arrived tasks will not be removed from RPU surface after finishing their execution, and they stay on RPU as far as possible. Therefore if these finished tasks are recurred in the future, we can reuse the existing configuration of the recurred tasks and the reconfiguration overhead will be eliminated. figure1b shows an example of this situation.

Although maintaining the tasks on RPU increases the probability of task reusing and decreases the reconfiguration overhead, it is not possible to maintain all arrived tasks on the RPU. Therefore the arriving tasks are divided into two groups: significant tasks and non-significant tasks, called most significant (MS) tasks and least significant (LS) tasks respectively. MS tasks are preserved on the RPU after their completion, while LS tasks are removed.

In fact, the existing tasks on the RPU are divided into three types at any given time: (1) *running tasks* which are currently executing on the RPU, (2) *scheduled tasks* which have been scheduled but their execution have not been started yet, (3) *preserved tasks* which their execution have been completed.

Also, RPU consist three different areas at any given time: (1) *free area* in which there is no task, (2) *occupied area* which includes the running tasks and scheduled tasks, (3) *preserved area* which includes the preserved tasks.

Figure 2 indicates an example of free, occupied and preserved areas on a RPU on time t_1, if T_2 and T_5 are MS tasks.



(a) (b)

Figure 1. An example of task scheduling and placement (a) without reconfiguration reusing and (b) with reconfiguration reusing

Considering the preservation may increase the fragmentation on RPU surface. In order to prevent RPU surface fragmentation, we partition RPU surface and put together the MS tasks in a partition. LS tasks are placed in another partition. The size of partitions may be fixed or dynamically be changed during run-time, in two different schemes which will be studied in section 5.

An issue that should be considered is that when the workload (number of arriving tasks per time unit) increases, preserving MS tasks on the RPU's MS partition may increase the probability of rejection of the new arriving tasks because the free area is reduced. In order to prevent task rejection in such situations, a technique is used for task replacement management which makes us able to replace preserved tasks by newly arrived task. In fact, the scheduling algorithm decreases the overall execution time of the tasks while preserving the rejection ratio in low value.

According to above description, the proposed algorithm includes four main parts: 1) task grouping, 2) task scheduling and placement, 3) task replacement management, and 4) dynamic partition resizing (only for dynamically adapted scenario).

These parts are described in the rest of this section in details.

4.1. Task Grouping

As described before the arriving tasks are divided into two groups: MS and LS tasks. We define a new significance function for each task called SG_i which includes following components:

1) The probability of task recurrence in the future; this parameter is denoted by P_i.

2) Reconfiguration overhead: the ratio of reconfiguration time to execution time of a task ($\frac{r_i}{er_i}$); this parameter is denoted by c.

3) Width ratio: the ratio of task width to partition width ($\frac{W_i}{W_P}$); this parameter is denoted by WR_i. The parameter W_P will be described in this section.

4) Reconfiguration speedup: the ratio of CPU execution time for task to total RPU execution time for task includes reconfiguration time and execution time for task($\frac{ec_i}{er_i + r_i}$); this parameter denotes by S_i.

Parameter P_i can be calculated using Poisson probability distribution function. According to Poisson distribution function, the probability of arriving k tasks in next time interval Δt is calculated as following:

$$P_{\Delta t}(k) = \frac{\lambda^{k} \times e^{-\lambda}}{k!} \quad (2)$$

where, λ is the average number of arrived tasks in past time interval Δt.

Thus, the probability of recurrence, P_i, for task T_i, is the probability of arriving one instance of task T_i in next time interval Δt, and is calculated as following:

$$P_{\Delta t}(1) = \frac{\lambda_i^{1} \times e^{-\lambda_i}}{1} = \lambda_i \times e^{-\lambda_i} = P_i \quad (3)$$

where, λ_i is the average number of arrived instances of task T_i in past time interval Δt.

The significance function of a task, SG_i, is a Boolean parameter which can be determined as following:

$$SG_i = (P_i \geq k_1) \wedge (O_i \geq k_2) \wedge (S_i \geq k_3) \wedge (WR_i \leq k_3) \quad (4)$$

where k_1, k_2, k_3 and k_4 are the thresholds between 0 and 1; when the value of SG_i is '1', the task is considered as significant task and vice versa.

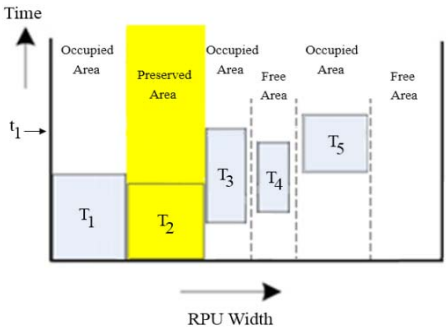


Figure 2. An example of RPU surface fragmentation

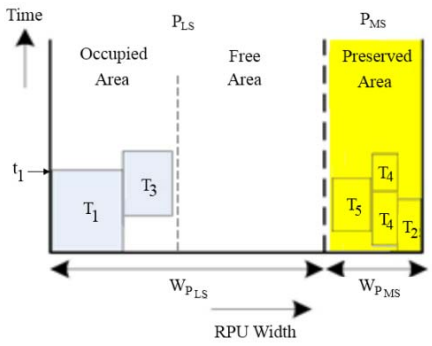


Figure 3. Total scheme of RPU partitioning

Equation (4) means that the task will be significant only if it has large probability of recurrence and it has large reconfiguration overhead and it has large gain for running on RPU and it consumes lower RPU area. Consequently, significant tasks are eligible for preserving on the RPU after their completion. Preserving significant tasks may eliminate the reconfiguration overhead for repetitive and low area consumer, high gain task for running on RPU.

A task list is constructed which maintains the history of arriving tasks including: the number of task recurrence along with recurrence time. When a new task arrives at the system, SG_i is calculated for newly arrived task using the task list information.

As described before, in this paper we consider two partitions for the RPU surface: P_{LS} and P_{MS} indicate portions for most significant task and low significant task respectively. The width of each partition can be found according to α, which is defined by following equation:

$$\alpha = \frac{W_{MS}}{W} \quad (5)$$

where W_{MS} and W denote the width of the most significant partition and width of RPU respectively. figure 3 shows the partitioning scheme. Partition P_{LS} is dedicated to place the LS tasks and partition P_{MS} is dedicated to place the MS tasks.

The partitions may be fixed in our first scheme and can be resized dynamically according to the conditions of input tasks in our second scheme. The details of partition resizing are explained in section 4.4.

4.2. Task Scheduling and Placement

We consider scanning-based placement method for task placement on the RPU. The tasks are placed from left to right in partition P_{LS}. In contrast, the tasks are placed from right to left in partition P_{MS}. This facilitates partition resizing which will be described in section 4.4.

In what follows, we briefly summarize the scheduling and placement algorithm.

Arrived tasks may be categorized into LS and MS groups. So there are two procedures when a task arrives.

a. Scheduling and placement of LS tasks

Firstly the free area of P_{LS} are checked by stuffing method and the new task will be placed in free area if it is possible. If not, resizing condition is checked (only in partition resizing scheme), and if P_{LS} can be expanded as large as the required area for newly arrived task, it will be placed in the resized partition. Finally if there will not sufficient area in the partition P_{LS}, even with resizing, the newly arrived task will be rejected from RPU. It should be noted that each task will be removed from P_{LS} as soon as its execution be finished, and will send to CPU for execution.

b. Scheduling and placement of MS tasks

There is a startup period for MS partition, in which P_{MS} is not full and has free area yet. During this period, the tasks are placed in RPU but will not remove after finishing. So P_{MS} will be full after transient time which is called startup period MS partition enters to steady state period, by ending the startup period. In steady state period, when a new significant

task arrives, firstly scheduler checks the existing task, if there is a copy of task in P_{MS} and it has preserved, the new arrived task be scheduled on the existing preserved task without paying any reconfiguration overhead.

If the existing instance of new task is running, then the finish time (f_j) of running instance is calculated according to the start time and execution time of the task ($f_j = s_j + er_j$). Also, the latest start time (LST) of new task is calculated regarding its deadline ($LST_i = d_i - er_i$). If the LST of new task is greater than the finish time of running instance, the new task will be scheduled immediately after the finish time of running instance and reconfiguration time of the new task will be eliminated.

If two above states have not occurred, the feasibility of partition expansion (resizing) will be examined. If partition expansion is allowed, then the partition is resized and new task may be placed on newly emerged area.

If partition expansion is not allowed or new task cannot be accommodated into newly emerged area, one of preserved tasks in partition P_{MS} is selected to be replaced by newly arrived task. The details of task replacement are explained in section 4.3. Finally if task replacement is not possible, the new task will be rejected.

All the lists which maintain occupied, preserved and free areas are updated at each event (task arrival time and task completion time).

4.3. Replacement Policy

As experessed in section 4.2 when there is not enough free space on P_{MS} to place a new task, one preserved task should be selected from preserved area to be replaced by newly arrived task.

Two different policies can be used: *best-fit* policy and *least probability of recurrence* (*LPR*) policy. In *best-fit* policy, the smallest preserved task which can accommodate newly arrived task is selected for replacement. In fact, *best-fit* policy aims at reducing the fragmentation on RPU surface. In *LPR* policy, the task which has the smallest probability of recurrence and is large enough to accommodate the new task is selected for replacement. In fact, the task which has the smallest probability of recurrence is the least recently recurred task, because the probability of recurrence, P_i , depends on the number of task recurrence in a specified time period.

4.4. Dynamic Resizing of RPU Partitions

Partition resizing may be a good solution to prevent task rejection when the workload is high. Partition resizing is to expand the size (width) of the partition provided that there is excess free area on adjacent partition. We consider a parameter, called *partition utilization* (*PU*), for each partition as following:

$$PU_i = \frac{W_{running}}{W_{P_i}} \tag{6}$$

where $W_{ruuning}$ and W_P are total width of occupied area, by running task in the partition and total width of the partition, respectively. The less the value of PU_i, the more there is free area on the partition P_i.

Resizing of partitions P_{LS} and P_{MS} are allowable (feasible) only if the other partition utilization is lower than a threshold that is considered 0.6 in this paper.

Figure 4 shows an example of partition resizing in which partition P_{MS} has been expanded. It should be noted that by reaching MS to steady state, it should be full, and after that there may be a free space only for short times that a task replaced by a smaller task during replacement procedure.

Partition resizing makes us able to change the virtual boundaries dynamically. This can decrease the probability of task rejection when the workload increases or when the task recurrence is low in an application. In section 5 we evaluate partitioning of RPU when there is fixed size partition and also compare the results with an adaptive resizing which described in this section.

5. Performance Evaluation

To evaluate the performance of partitioning of RPU and newly proposed metrics for grouping arrived tasks, SG_i , we developed a discrete time simulation framework in Matlab. It has been carried out on a 2 GHz Core 2 Duo computer. The simulated device consists of 96*64=6144 reconfigurable units (RCUs), which corresponds to xilinx's XCV1000 FPGA.

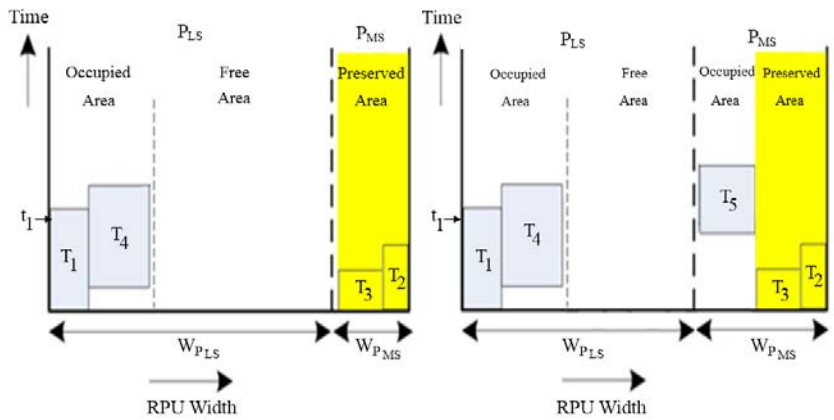


Figure 4. An example of partition resizing (a) before expansion of P_{MS}, (b) after expansion of P_{MS}

We have conducted our experiments on groups of input tasks called task sets. Each task set includes 500 synthetic tasks. All synthetic tasks have randomly generated parameters. The probability distribution of task parameters are based on real tasks in real world applications, such as DCT [28] and FFT, DWT transformations and triple-DES crypto-algorithm [29].

Other parameters of the generated task sets are summarized in table 1.

To study the performance of reusing and resizing in different conditions, we consider three input parameters. Firstly α which was defined in Equation (5). The second one is recurrence property of the tasks. So a parameter called *Repetition Rate* (*RR*) is considered for each task set as following:

$$RR = \frac{number\ of\ repetitive\ tasks\ in\ a\ task\ set}{total\ number\ of\ tasks\ in\ a\ task\ set} \times 100 \quad (7)$$

In fact, parameter *RR* indicates the percentage of repetitive tasks in each task set.

The third one is workload which is defined as follows:

$$WL_{\Delta T} = \frac{\sum_{T_i \in \Delta T} w_i \times er_i}{W \times \Delta T} \quad (8)$$

where W is the total width of RPU and w_i, er_i were defined in Equation (1). In fact $WL_{\Delta T}$ represents the volume of the occupied area and time by arriving tasks in time interval ΔT.

The parameters that we have derived by simulation include total execution time (TET) of task sets, total rejection ratio (TRR) which indicates the ratio of tasks rejection from RPU, and speedup. Speedup shows the improvement in speed for reconfigurable system consists of a RPU and a CPU, comparing a non-reconfigurable system consists only a CPU.

Table 1. Probability distribution of the parameters of the generated tasks

Width (w_i)	Execution Time on RPU (er_i)	Slack ($d_i - a_i - er_i - r_i$)	Reconfiguration to Execution Ratio (r_i / e_i)	Speedup (S_i)
Uniform Distribution in interval [2 , 40]	Uniform Distribution in interval [5 , 100]	Uniform Distribution in interval [1 , 50]	Uniform Distribution in interval [0.1 , 0.5]	Uniform Distribution in interval [3 , 5]

5.1. Simulation Results

In the first step of performance evaluation of reusing technique we consider two partitions with a fixed size ratio of MS partition in the RPU. Figure 5 shows the TET, TRR and speedup, in terms of size ratio (α) of fixed MS partition, with several RR values. These curves show that if the value of α has not been chosen correctly, the performance of the system, will be drop strongly. For example if the RR of the incoming tasks were high and a small value has been chosen for α, then most of the MS tasks will be rejected to the CPU and the reusing approach will not be fully exploited, and on the other hand if this size ratio of fixed MS partition considered high for a small value of the RR, then amount of rejected LS tasks will be grown gradually and the performance of the system will be suffered from an inappropriate allocation of resources.

Although the figures show that there is a tradeoff in the curves and one can find an optimum value for each RR but since the behavior of the incoming tasks may be dynamically changed, designing by a fixed size ratio for MS partition (fixed α) is not a good solution for achieving the optimum performance in both MS and LS partitions. For this reason an adaptive resizing on partitions of the RPU width has been used in this paper and figure 6 and 7 shows the TET, TRR and Speedup curves for this state by considering several parameters. In figure 6 the repetitive ratio has been considered 30% and initialized size ratio for the MS are 25%, 50% and 75%. In figure 7 the initialized size ratio for MS has been considered 50% and the RR are 20%, 40% and 60%. In both cases the workload of tasks has been derived from Equation (7).

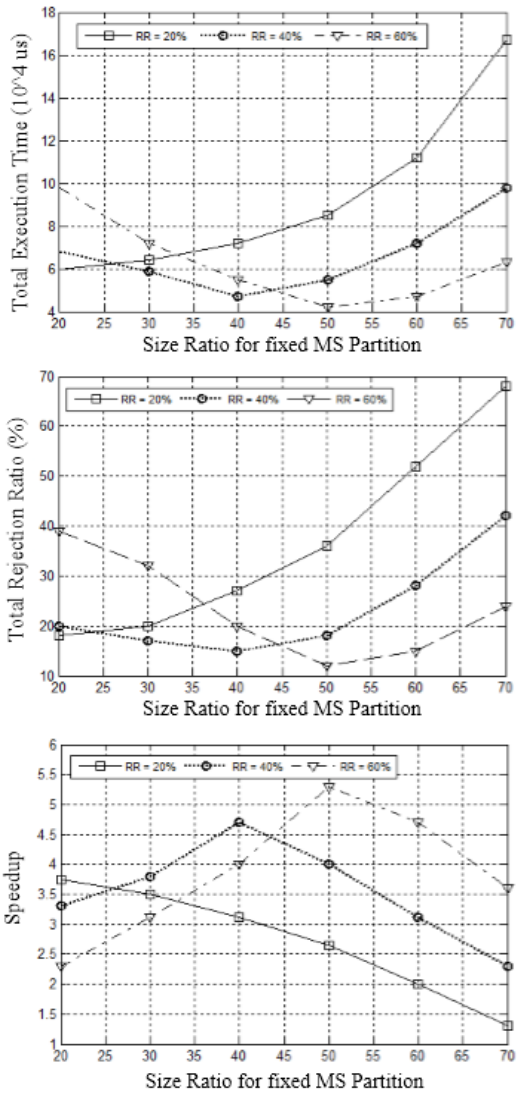


Figure 5. TET, TRR and speedup curves for the fixed size ratio of MS partition by several repetitive ratio

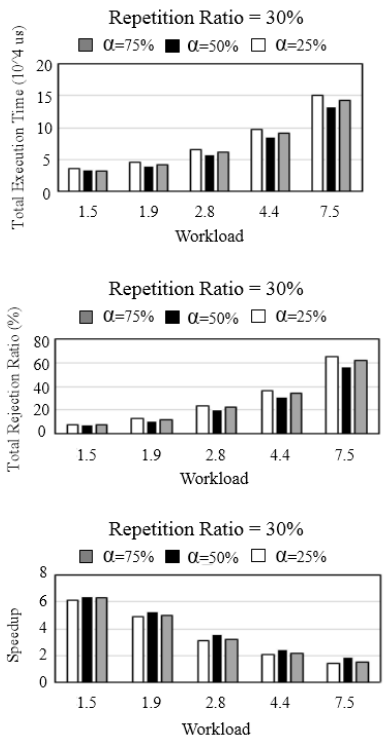


Figure 6. TET, TRR and Speedup curves by reusing and resizing approach in terms of workload with several MS size ratio

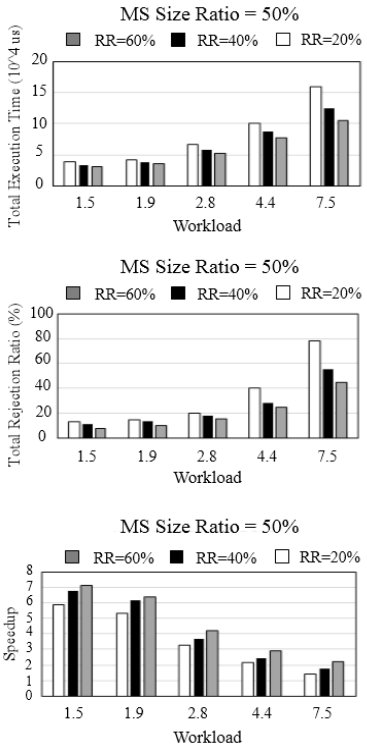


Figure 7. TET, TRR and Speedup curves by reusing and resizing approach in terms of workload with several Repetitive ratio

Figure 6 shows that in a low workload, choosing a correct MS size ratio will not affect very intensively, because there is always some free space in the RPU for resizing and the performance of several MS size ratio are approximately equal. But when the workload grows then tasks will rapidly fill RPU.

In such situation choosing a suitable initialized MS size ratio will be important. The best initial value for α depends on recurrence ratio of incoming tasks, RR. This phenomena shows in figure 6 when the workload is high, the rejection ratio of LS tasks will be grown and the total performance of the system will be degraded, because both partitions have been full and resizing is not possible. Figure 7 also shows the effectiveness of RR on the system performance. It can be seen that in a high workload by using an enough space for the MS tasks for large RRs, a good performance can be achieved. In both figure 6 and 7 the speedup of the system will be degraded by increasing the workload, because the rejected tasks to the CPU will be increased and less tasks can uses the RPU capability to have a small execution time.

Figure 8 shows the comparison of three approach partitioning in the RPU in total execution time, one partition, two fixed partitions with reusing, and two resizable partitions with reusing. The good behavior of reusing and resizing approach has been depicted in this chart. Figure 8 shows, when RR is 60%, there is up to 35% improvement in the overall execution time of tasks for scheduling with two resizable partitions and reusing, comparing one partition RPU (no partitioning scheme); and there is 20% improvement in the overall execution time of tasks for scheduling with two resizable partitions and reusing, comparing two fixed partitions with reusing. These values will respectively be 25% and 14% when RR is 40% .

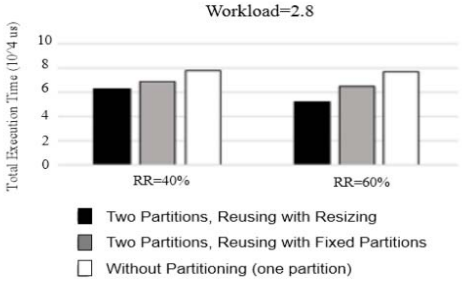


Figure 8. Comparison of three approach, one partition, two partitions with reusing and fixed partitions, and two partitions with reusing and resizing

So with enough percent of recurrence ratio of incoming task, reusing and resizing approach can decrease the overhead time of reconfiguring of the task and reduce the total execution time consequently.

6. Conclusion

In this paper, the problem of on-line scheduling for RC systems is addressed. We consider two virtual partitions for RPU surface and schedules arriving tasks on different partitions according to their significance. The related works are reviewed and new significant criteria for task reusing have been presented. The significance of the input tasks depends on their probability of recurrence in the near future, their reconfiguration overhead, their size and their speedup. The significant tasks are maintained on the RPU after their completion as far as possible. Therefore they are reused

when they recur in the future. Also, dynamic partition resizing and task replacement management makes the proposed algorithm stronger and more flexible.

Simulation results show these benefits of the proposed algorithm. It has been shown that there is a tradeoff for α in fixed partitions scheme, and also how resizing partitions can improve the performance.

Future work can be concentrated on studying the effect of replacement policies and changing the resizing strategies. The other area for future work is extending the current work to 2D task model.

References

[1] C. Kao, "Performance-Oriented Partitioning for Task Scheduling of Parallel Reconfigurable Architectures," *IEEE Trans. Parallel and Distributed Systems*, vol. 25, no. 1, pp. 1-10, 2014.

[2] P. Saha, and T. El-Ghazawi, "A Methodology for Automating Co-Scheduling for Reconfigurable Computing Systems," *Proc. IEEE/ACM Intl Conf. Formal Methods and Models for Codesign*, pp. 159-168, 2007.

[3] A. Agne, M. Happe, A. Keller, E. Lubbers, B. Plattner, M. Platzner, and C. Plessl, "Recon OS-An Operating System Approach for Reconfigurable Computing," *IEEE Micro*, vol. 34, no. 3, pp. 60-71, 2014.

[4] Q. Deng, S. Wei, H. Xu, Y. Han, and G. Yu, "A Reconfigurable RTOS with HW/SW Co-scheduling for SOPC," *Proc. Second Intl Conf. Embedded Software and Systems*, pp. 60-66, 2005.

[5] M. M. Bassiri, and H. S. Shahhoseini, "Configuration Reusing in On-Line Task Scheduling for Reconfigurable Computing Systems," *Journal of Computer Science and Technology*, vol. 26, no. 3, pp 463-473, 2011.

[6] K. Bazargan, R. Kastner, and M. Sarrafzadeh, "Fast Template Placement for Reconfigurable Computing Systems," *IEEE Design and Test of Computer Journal*, vol. 17, no. 1, pp. 68-83, 2000.

[7] Q. H. Khuat, D. Chillet, and M. Hubner, "Considering Reconfiguration Overhead in Scheduling of Dependent Tasks on 2D Reconfigurable FPGA," *Proc. IEEE Intl Conf. Adaptive Hardware and Systems*, pp.1-8, 2014.

[8] T. Marescaux, A. Bartic, V. Dideriek, S. Vernalde, and R. Lauwereins, "Interconnection Networks Enable Fine-Grain Dynamic Multi-tasking on FPGAs," *Proc. IEEE Intl Conf. Field-Programmable Logic and Applications*, pp. 795-805, 2002.

[9] Xilinx, "Field Programmable Gate Arrays," December 2002.

[10] S. Banerjee, E. Bozorgzadeh, and N. Dutt, "Integrating Physical Constraints in HW-SW Partitioning for Architectures With Partial Dynamic Reconfiguration," *IEEE Trans. Very Large Scale Integration*, vol. 14, no. 11, pp. 1189-1209, 2006.

[11] I. Belaid, F. Muller, and M. Benjemaa, "Optimal Static Scheduling of Real-time Dependent Tasks on Reconfigurable Hardware Devices," *Proc. IEEE/ACM Intl Conf. Formal Methods and Models for Codesign*, pp. 1-6, 2011.

[12] M. Huang, H. Simmler, and P. Saha and T. El-Ghazawi, "Hardware Task Scheduling Optimizations for Reconfigurable Computing," *Proc. IEEE Intl workshop on High-Performance Reconfigurable Computing Technology and Applications*, pp. 1-10, 2008.

[13] W. Hu, and C. Wang, "A Novel Approach for Finding Candidate Locations for On-line FPGA Placement," *Proc. IEEE Intl Conf. Field-Programmable Logic and Applications*, pp. 2509-2515, 2010.

[14] X. G. Zhou, Y. Wang, X. Z. Huang, and C. L. Peng, "On-line Scheduling of Real-time Tasks for Reconfigurable Computing System," *Proc. IEEE Intl Conf. Field-Programmable Logic and Applications*, pp. 25-31, 2006.

[15] K. Danne, and M. Platzner, "A Heuristic Approach to Schedule Periodic Real-Time Tasks on Reconfigurable Hardware," *Proc. IEEE Intl Conf. Field Programmable Logic and Applications*, pp. 568-573, 2005.

[16] A. Ahmadinia, C. Bobda, and J. Teich, "A Dynamic Scheduling and Placement Algorithm for Reconfigurable Hardware," *Proc. IEEE Intl Conf. Field-Programmable Logic and Applications*, pp. 125-139, 2004.

[17] T.Y. Lee, N. Y. Lin, W. C. Chen, and H. Wu, "An Efficient Task Placement Method for Reconfigurable FPGA Systems," *Proc. IEEE Intl Conf. Complex, Intelligent, and Software Intensive Systems*, pp. 451-455, 2013.

[18] A. Al-Wattar, S. Areibi, and F. Saffih, "Efficient On-line Hardware/Software Task Scheduling for Dynamic Run-Time Reconfigurable Systems," *Proc. IEEE Intl Symp. Parallel and Distributed Processing*, pp. 401-406, 2012.

[19] T. Marconi, Y. Lu, K. Bertels, and G. Gaydadjiev, "Online Hardware Task Scheduling and Placement Algorithm on Partially Reconfigurable Devices," *Proc. IEEE Intl Conf. Adaptive Hardware and Systems*, pp. 306-311, 2008.

[20] Y. Sheng, Y. Liu, R. Li, and X. Xiao, "A Communication-aware Scheduling Algorithm for Hardware Task Scheduling Model on FPGA-based Reconfigurable Systems," *Journal of Computers*, vol. 9, no. 2, pp. 2552-2558, 2014.

[21] T. Marconi, "Online Scheduling and Placement of Hardware Tasks with Multiple Variants on Dynamically Reconfigurable Field-programmable Gate Arrays," *Computers and Electrical Engineering Journal*, vol. 40, no. 4, pp. 1215-1237, 2014.

[22] S. Roman, H. Mecha, D. Mozos, and J. Septien, "Constant Complexity Scheduling for Hardware Multitasking in Two Dimensional Reconfigurable Field-programmable Gate Arrays," *Journal of IET Comput. Digit. And Tech*., vol. 2, no. 6, pp. 401-412, 2008.

[23] A. Hassanli, A. Khayatzadeh Mahani, H. S. Shahhoseini, and E. Teimori, "Queuing Analysis for Reconfigurable," *Proc. IEEE Intl Workshop on Power Electronics and Intelligent Transportation System*, pp. 284-288, 2008.

[24] J. Cui, Z. Gu, W. Liu, and Q. Deng, "An Efficient Algorithm for On-line Soft Real-time Task Placement on Reconfigurable Hardware Devices," *Proc. IEEE Intl Symp. on Object and Component-Oriented Real-Time Distributed Computing*, pp. 321-328, 2007.

[25] F. Dittmann, and S. Frank, "Caching in Real-time Reconfiguration Port Scheduling," *Proc. IEEE Intl Conf. Field-Programmable Logic*, pp.740-744, 2007.

[26] L. Liang, X. Zhou, Y. Wang, and C. Peng, "On-line Hybrid Task Scheduling in Reconfigurable Systems," *Proc. IEEE Intl Conf. Computer Supported Cooperative Work in Design*, pp. 1072-1077, 2007.

[27] J. Cui, Q. Deng, X. He, and Z. Gu, "An Efficient Algorithm for Online Management of 2D Area of Partially Reconfigurable FPGAs," *Proc. IEEE Intl Conf. Design, Automation and Test in Europe*, pp. 129-134, 2007.

[28] Xilinx Inc., Virtex Core Generator, December 2002.

[29] Amphion Semiconductor Group, http://www.amphion. com, May 2003.

Hadi Shahriar Shahhoseini received B.S. degree in electrical engineering from University of Tehran, in 1990, M.S. degree in electrical engineering from Azad University of Tehran in 1994, and Ph.D. degree in electrical engineering from Iran University of Science and Technology, in 1999. He is an associate professor of the electrical engineering department in Iran University of Science and Technology. His areas of research include networking, supercomputing and reconfigurable computing. More than 150 papers have been published from his research works in scientific journals and conference proceedings. He is an executive committee member of IEEE TCSC and serves IEEE TCSC as regional coordinator in middle-East Countries.

E-mail: hshsh@iust.ac.ir

Maisam Mansub Bassiri received the B.S. and M.S. degrees in electrical engineering from Sharif University of Technology (Iran) and Amirkabir University of Technology (Iran) in 2002 and 2004, respectively. Also, he received Ph.D. degree in electrical engineering from Iran University of Science and Technology in 2010 and now he is an assistant professor in Azad University of Tehran. His research interests are in the field of parallel processing, embedded system design and reconfigurable computing. Now, his current research focuses on HW/SW co-design and resource management in reconfigurable computing systems. Also he has published several papers in his research fields so far.

E-mail: basiri@ee.iust.ac.ir

Seyed Mehdi Mohtavipour received the B.S. and M.S. in electrical engineering from University of Guilan and Iran University of Science and Technology (IUST) in 2012 and 2014, respectively. He is now Ph.D. student at Iran University of Science and Technology (IUST) and his research interests include intelligent transportation systems and reconfigurable computing.

E-mail: mehdi_mohtavipour@elec.iust.ac.ir

Paper Handling Data:

Submitted: 12.04.2010
Received in revised form: 06.09.2012
Accepted: 08.11.2014
Corresponding author: Dr. Hadi Shahriar Shahhoseini, Electrical Engineering Department, Iran University of Science and Technology, Tehran, Iran.