
MAISAM MANSUB BASSIRI AND HADI SHAHRJAR SHAHHOSEINI

Department of Electrical Engineering
Iran University of Science and Technology
Narmak, 16846-13114, Iran
E-mail: hshsh@iust.ac.ir

Reconfigurable computing systems allow executing tasks in a true multitasking manner. Such systems share the reconfigurable device and processing unit as computing resources which leads to highly dynamic allocation situations. To manage such systems at runtime, a reconfigurable operating system is needed. The main part of this operating system is resource management unit which performs HW/SW partitioning, co-scheduling and placement of hardware tasks at run-time. In this paper, we present a technique for on-line integrated HW/SW partitioning and co-scheduling. We focus on on-line, real-time and non-preemptive systems. The main characteristics of our method are future-awareness scheduling and strong nexus between partitioning, scheduling and placement. A large variety of experiments have been conducted on the proposed algorithm using synthetic and real tasks. Obtained results show considerable benefits of this algorithm.

Keywords: reconfigurable computing, HW/SW partitioning, co-scheduling, placement, real-time tasks

1. INTRODUCTION

Reconfigurable computing (RC) systems comprise one or several reconfigurable processing unit(s) (RPU) along with a CPU. Such systems may be divided into two categories, depending on the relation between RPU(s) and CPU. First category includes the RC systems in which CPU doesn’t intervene in computation and only manages the reconfiguration of RPU(s). Second category includes the RC systems in which CPU contributes the computation and the synergy between CPU and RPU improve the computing performance. Saha et al. in [1] call the first category as reconfigurable hardware (RH) systems and second category as reconfigurable computing (RC) systems. RPU which may be a modern FPGA can be reconfigured at run-time and support partial reconfigurability which makes us able to change the reconfiguration of part of FPGA dynamically without disturbing other parts. Hence they have high flexibility along with high performance. Run-time reconfigurability and partial reconfigurability of FPGAs allow us to execute tasks in a true multitasking manner.

In this paper we target the second category of RC systems in which synergy between CPU and RPU improves the computing performance. As in the first category, all the tasks are executed by the RPU, therefore HW/SW partitioning of the tasks is not meaningful. In contrast, in the second category, HW/SW partitioning of the tasks is an important step in...
computing. While using RC systems can increase the computing performance, it also leads to complex allocation situations for dynamic task sets. This clearly asks for well-defined system services that help to efficiently design applications. Such a set of system services forms a reconfigurable operating system (OS) [2, 3]. From the designer’s point of view, an operating system is an abstraction layer in design process that hides the details of underlying hardware. The benefits of using this abstraction layer are increased design productivity, portability and resource utilization. Of course these benefits are paid for by area overhead and computation time overhead. This work deals with the resource management part of a reconfigurable OS. In particular, we focus on the problem of on-line HW/SW partitioning and co-scheduling the tasks to the RPU and CPU in a RC system.

In the rest of this paper, section 2 discusses the modeling of the RC system, RPU, the tasks and the operating system. Section 3 reviews the related work. In section 4 we discuss on-line integrated HW/SW partitioning and co-scheduling and present our algorithm. In section 5, complexity analysis of the algorithm is presented. Section 6 presents the experimental evaluation of the algorithm and finally, section 7 concludes the paper.

2. PROBLEM MODELING

In this section, we first present the modeling of RPU and the tasks. Then we introduce our system architecture and OS model. Finally, we discuss HW/SW partitioning and co-scheduling problem.

2.1 RPU and Task Models

A task may be a function synthesized to digital circuit (hardware form) that can be programmed onto the reconfigurable device (RPU) or implemented by software code (software form) that can be run on a CPU. Hardware form of a task has a size and a shape. The size gives the area requirement of the task in reconfigurable units. We assume all task shapes to be rectangle and the execution time of tasks are known in advance. Generally, each task is defined as a 6-tuple $T_i = (w_i, h_i, e^{s}_i, e^{h}_i, a_i, d_i)$ where $w_i$, $h_i$, $e^{s}_i$, $e^{h}_i$, $a_i$ and $d_i$ denote width, height, software (SW) execution time, hardware (HW) execution time, arrival time and deadline of the task, respectively. The reconfiguration time of the task can be considered as a part of its HW execution time. Also, we define parameter task interval ($TI$) for a task as the difference between deadline and arrival time of the task ($TI_i = d_i - a_i$).

Tasks can be real-time or non real-time. Deadline ($d_i$) 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 may be represented as a directed task graph. In this work we focus on real-time and independent tasks. Also, the task execution can be preemptive or non-preemptive. Although preemption is a very useful technique that yields efficient and fast on-line scheduling algorithm, we believe that it is too expensive for current reconfigurable technology, because preemption and resuming task, requires heavy (large) context switches and needs additional external memory. Hence, in this work we focus on a non-preemptive task model i.e. once a task is loaded onto the device, it runs to completion.

The mapping of tasks to a RPU strongly depends on the area model of RPU. Therefore it is necessary to describe our RPU area model. In general, there are four area models.
The flexible 2D area model (Fig. 1 (a)) allows to allocate rectangular tasks anywhere on the 2D reconfigurable surface. This model has been used by many researchers [4-6]. The advantage of this model is high device utilization but on the other hand high flexibility leads to difficult scheduling and placement algorithms. Another model is partitioned 2D model (Fig. 1 (b)) where the reconfigurable surface is split into a statically-fixed number of allocation sites, so called blocks [7, 8]. Each block can accommodate at most one task at a time. Such a model facilitates the scheduling and placement but causes internal fragmentation. In the flexible 1D area model shown in Fig. 1 (c), 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]. This model leads to low complexity placement problem. On the other hand, it suffers from both internal and external fragmentation. Finally, there is partitioned 1D area model as shown in Fig. 1 (d). This model combines the characteristics of a partitioned model and 1D model. Again the disadvantage lies in the potentially high internal fragmentation.

In this paper, we address the problem of on-line HW/SW partitioning and co-scheduling of real-time tasks to the 1D area model. The tasks have known execution times and there is no dependency among them. Such assumptions are the same as used in related research works. There are large variety of applications which can exploit our proposed algorithm including: reconfigurable co-processor implementation, image and video processing, cryptography, telecommunication and neural network implementation.

2.2 System Architecture and OS Model

Fig. 2 shows the target system architecture. As shown, the system comprises a CPU, RPU, shared memory and configuration memory. Arrived tasks are stored in a queue where they await further processing. As stated in section 1, Resource management is the most important part of the reconfigurable OS. The main parts of resource management unit are implemented by three modules: HW/SW partitioner and co-scheduler (PSU), placer and configuration loader. As stated in [2], main parts of the OS are implemented on CPU.
and the other parts are implemented on RPU. PSU module receives the input tasks and uses a HW/SW partitioning and co-scheduling algorithm to allocate a processing unit to each task and assign a start time to it. After partitioning, HW tasks are sent to HW queue and SW tasks are sent to SW queue. SW tasks are run on CPU one by one and HW tasks are placed on RPU by placer and configuration loader. Placer uses a placement algorithm to assign a well-suited location to each task. Configuration loader reads the configuration bitstream from configuration memory and loads it into the RPU. Loader module can only load one task at a time. The nexus of scheduling and placement is the main characteristic for the problem of mapping tasks to RPU. Since there is no guarantee that the placer finds a feasible placement for every task that has been selected for execution, therefore the partitioner and co-scheduler algorithm must incorporate the placement algorithm.

Data communication between HW tasks and SW tasks is performed via shared memory. HW tasks use predefined communication interface to communicate each other and access special resources such as block RAMs. Configuration bitstream of HW tasks are stored in configuration memory and loader can access it.

2.3 HW/SW Partitioning and Co-scheduling Problem Definition

In this section, we define the problem of HW/SW partitioning, co-scheduling and placement. As stated, in this work we use 1D area model for device area. Therefore the location of each task $T_i$ on the device surface is indexed with $x_i$ which is the horizontal position of bottom-left of rectangular task ($T_i$). We can define the HW/SW partitioning, placement and co-scheduling for a task as following.

**Definition 1** HW/SW partitioning $\text{PART}(T, TH, TS)$ partitions the task set $T$ into two subsets $TH$ and $TS$ such that:

$TH \cap TS = \emptyset,$

$TH \cup TS = T.$
Definition 2 A placement $P(T_i, T_c)$ for a HW task $T_i$ and a set of currently placed tasks $T_c$, is a placement <$x_i$> that satisfies following constraints in 1D area model:

$$x_i + w_i \leq W,$$

$$h_i \leq H,$$

$$\forall T_j \in T_c: (x_i + w_i \leq x_j) \lor (x_j + w_j \leq x_i).$$

$H$ and $W$ are height and width of the device surface, respectively. In fact, the hardware tasks must be placed in such a way that they don’t overlap with the device boundaries and other currently placed tasks.

Definition 3 Co-scheduling assigns a starting time $s_i$ to each task $T_i \in T$ such that:

$$\forall T_j \in T, T_j \neq T_i:$$

$$\begin{cases} (s_i + e_i^e \leq s_j) \lor (s_j + e_j^e \leq s_i), & \text{if } T_j, T_j \in T_S \\ (x_i + w_i \leq x_j) \lor (x_j + w_j \leq x_i) \lor (s_i + e_i^h \leq s_j) \lor (s_j + e_j^h \leq s_i), & \text{if } T_i, T_j \in T_H. \end{cases}$$

For real-time tasks, the following equations should be satisfied too, in co-scheduling:

$$\begin{cases} s_i + e_i^e \leq d_i, & T_i \in T_S \\ s_i + e_i^h \leq d_i, & T_i \in T_H. \end{cases}$$

At any given time, Allocation ($Ac$) includes the currently partitioned, scheduled and placed tasks. In an on-line scenario, new tasks arrive during the system’s runtime. For each arriving task $T_i$, or arriving task set $T$, HW/SW partitioner has to partition the arrived tasks into HW or SW; and scheduler has to find feasible start times. If scheduler and placer cannot find a feasible placement and start time for the arrived task that satisfy the mentioned constraints, the arrived task will be rejected. In such real-time systems, it is assumed that when a task is rejected, it is referred to the resource out of the system for execution. The goal of HW/SW partitioning and co-scheduling in target RC systems is to minimize the task rejection ratio (TRR) in the system. Task rejection ratio is the ratio of number of rejected tasks to the total number of arrived tasks.

2.4 Task Placement

According to placement definition stated in previous section, there exist two different approaches for task placement: scanning and management of free/occupied spaces.

The main advantage of the scanning techniques is that they 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. The drawback of this approach is its high run-time complexity for large device area. In 1D area model, the run-time complexity of scanning technique is acceptable. In fact, in the worst case, $W$ (the width of the RPU in reconfigurable units) potential placements have to be considered.

In free/occupied space management techniques, the list of free/occupied spaces is maintained and is updated in each event. In these techniques, for placement a task, only the
free spaces are searched. Some of space management techniques such as MER (maximal empty rectangle) management [5] are recognition-complete and some of them are not.

Generally a placer contains two sub-functions: the status holder and the fitter. The status holder maintains the status of reconfigurable units. Whenever a task is placed or removed, status holder updates the list of free (occupied) reconfigurable units. The fitter selects a site among the several feasible sites for task placement. Several fitting strategies can be applied. Best-fit for example, chooses the smallest free space which is big enough to accommodate the task. First-fit strategy selects the first free space which can accommodate the task.

3. RELATED WORK

So far, many important works such as [10-14] have been carried out on HW/SW partitioning and co-scheduling in embedded computing (EC) systems with fixed hardware resources such as ASICs. As stated before, several features such as run-time reconfigurability (RTR) or partial reconfigurability (PR) which exist in the RC systems have not been addressed in those works. Other related work may be divided into following three categories:

(1) First category includes the works such as [15-20] which address HW/SW partitioning and co-scheduling in RC systems. But their focus is on off-line applications in which task graph characteristics are known in advance. For example, Banerjee et al. in [19] present a placement-aware algorithm for integrated HW/SW partitioning and scheduling in off-line applications. In the other work [17], Haubelt et al. present a system-level approach for HW/SW partitioning and scheduling in multi-processor and multi-FPGA RC system using multi-objective evolutionary algorithm. Also, Noguera et al. in [21] present a semi-online Algorithm that performs off-line partitioning and on-line scheduling in a multi-context RC system. Most of the off-line algorithms use evolutionary algorithms for partitioning and scheduling which are not usable in on-line applications.

(2) Second category includes the works such as [22-30] which address on-line scheduling in RH systems. Although some of those are good and efficient algorithms, the synergy between CPU and RPU is not considered in computing and therefore HW/SW partitioning is not addressed in these works. Some of these works such as [23] focuses on real-time and preemptive tasks. But most of them focus on real-time and non-preemptive tasks. Zhou et al. in [22] present a technique that uses time window to find feasible location for placing on-line real-time tasks. In fact, they have improved the stuffing algorithm for scheduling. Ahmadinia et al. in [24] present an integrated scheduling and placement algorithm in which the FPGA is divided into slots and the arriving tasks are placed inside one of the slots depending on their execution end time. Moreover the width of the slots is to be varied during runtime in order to improve the quality of placement. In [26], two techniques, horizon and stuffing, are presented by steiger, walder and platzner. Both proposed techniques use reservation list and free space list for scheduling. The stuffing method schedules tasks into arbitrary free rectangles that will exist in the future. But horizon scheduler can only append the tasks to the horizons and it doesn’t guarantee to find a feasible placement even though there is enough free space on the RPU.
Marconi et al. in [27] present intelligent stuffing method. They define alignment status of the free space segments and use it to make the correct decision on task placement position in order to maximize the free space area. In another work [30], Roman et al. have proposed a technique in which FPGA area is divided into four partitions of different sizes. 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.

(3) Third category includes the works such as [31-37] which address on-line HW/SW partitioning and co-scheduling in different RC systems. Pellizzoni and Caccamo in [31] present an on-line partitioning and scheduling algorithm based on task relocatability. They first, divide the arrived tasks into HW tasks and SW tasks randomly, and then improve the partitioning using task relocations at run-time. As task preemption does incur a large overhead, the authors consider some constraints in task relocation. In the other work [32], Streichert et al. present a system-level approach for on-line HW/SW partitioning and load balancing in the embedded systems consisting of networked HW/SW reconfigurable nodes. They also, assume that task relocation between SW nodes and HW nodes is possible. Fakhreddine et al. in [33] present an on-line HW/SW partitioning and scheduling algorithm for soft real-time applications with data dependency such as image processing. In this algorithm, after execution completion of each frame, task characteristics of the next frame are predicted and task allocation is performed based on estimated parameters. Obviously, this algorithm is only applicable on data-dependant applications. In [34], Ali and Das present an efficient heuristic algorithm for enforcing the schedulability of aperiodic hard real-time tasks arriving simultaneously with precedence constraints and individual deadlines in multi-FPGA RC systems. The proposed co-synthesis algorithm integrates partitioning and non-preemptive scheduling. Reconfigurable FPGAs are incrementally added when schedulability suffers in a uniprocessor system. Liang et al. in [35] have presented a combined scheduling algorithm which covers task allocation, placement and migration. In this method, scheduler uses decision function and reservation and utilization-first policy on allocation, placement and migration respectively. The main target of this method is improving the utilization of RPU and admitting more tasks to execute on RPU. In [36], Cui et al. present a novel fragmentation metric based on MERs that take into account probability distribution of width and height of future task arrivals instead of just the average value. In addition, they take into account the time axis to minimize the time-averaged area fragmentation (TAAF) during the execution time of the task being placed. They use TAAF to select the best feasible placement among the several placement choices.

4. HW/SW PARTITIONING AND CO-SCHEDULING

In an on-line and real-time RC system, real-time tasks arrive at arbitrary times and must be partitioned into HW or SW tasks and scheduled on RPU or CPU. In such systems, the goal is to minimize the task rejection ratio (TRR). In this section, we present our technique, for on-line HW/SW partitioning and co-scheduling in target RC system.
Fig. 3 shows the flowchart of the overall proposed algorithm. As shown in the figure, the main parts of the algorithm are as following,

(1) **Parameter estimation**: in this part of algorithm, calculation of some necessary parameters for HW scheduler is carried out. HW scheduler uses a predictive technique to perform future-aware high quality scheduling. Hence, it needs some important parameters which are calculated and updated by this part of algorithm. Detail information about this part will be presented in section 4.4.

(2) **Task prioritizing**: in on-line and real-time scenario, there is the possibility of arriving several tasks at a time. In such situations, the simultaneously arrived tasks must be prioritized and scheduled according to their priorities. This part of algorithm performs the task prioritizing function.

(3) **HW/SW partitioning**: this part performs initial HW/SW partitioning. After initial partitioning the tasks are referred to the assigned scheduling unit (HW scheduler or SW scheduler). In schedulers, the schedulability test of the tasks is carried out. If feasible scheduling is not found by the assigned scheduler, the task will be moved to the other partitioner (returned task). If the other scheduler can not find a feasible scheduling for the task too, then the task will be rejected.

(4) **HW scheduling and software scheduling**: these parts perform scheduling of the HW and SW tasks, respectively. The SW scheduling is performed in straightforward method. But HW scheduler estimates the requirements of the future tasks and uses them to make a correct decision for placement of current tasks.

We have integrated the on-line HW/SW partitioning and on-line scheduling algorithms. As stated in section 1, the nexus between HW/SW partitioning, scheduling and
placement is one of the main characteristics of our algorithm which guarantees finding feasible high quality solutions.

In the following sections, we describe the aforementioned parts of the algorithm in details.

4.1 Task Prioritizing

When two or more tasks arrive at the system simultaneously i.e. they have synchronous arrival time, the tasks must be prioritized. In some recent work such as [26], the task deadlines are considered as the main parameter for task prioritizing in real-time systems and EDF (earliest deadline first) policy is used for this purpose. But the deadline of the task, by itself, can not be a reasonable parameter for prioritizing. For illustration, let consider two tasks $T_1$ and $T_2$ as shown in Fig. 4. Those have the same arrival time but the deadline and execution time of the $T_2$ is smaller than those of $T_1$. Although the deadline of $T_2$ is smaller than the one of $T_1$, it is absolutely clear that the scheduling of $T_1$ is more urgent than $T_2$ because the available time interval for scheduling of $T_1$ is very small.

Fig. 4. Two tasks with the same arrival time, different execution times and different deadlines.

We define a new parameter, urgency, based on the Latest Start Time (LST) of the task and use it to prioritize the task.

$$LST_i = d_i - e_i^h$$
$$S_i = LST_i - a_i$$
$$U_i = 1/(LST_i - a_i) = 1/S_i$$

Where, $U_i$ denotes the urgency of the task and $S_i$ denotes the slack of the task ($T_i$). The tasks which have smaller LST, have higher urgency and higher priority.

In above example, $T_1$ and $T_2$ have the same arrival time but the latest start time (LST) of $T_1$ is smaller than $T_2$, therefore its priority is higher than $T_2$.

In the proposed algorithm, task prioritizing is performed based on task urgency. If two simultaneous tasks have the same value of urgency, the task which has the earliest deadline, will have the highest priority. In rare conditions that two simultaneous tasks have the same urgency and the same deadline, one of the tasks is considered as the more priority task, randomly. Therefore, task prioritizing doesn’t incur conflicts.

After urgency-based prioritizing of the tasks, the arrived tasks are selected according to their priorities (the highest priority is the first). Each selected task is partitioned into HW or SW and then scheduled as described in next sections.
4.2 HW/SW Partitioning

A trivial and straightforward method for HW/SW partitioning which has been addressed in some recent research works may be called *hardware-first* technique. In this technique, the arrived tasks are applied to the hardware at first, provided that enough free space exists on the RPU, otherwise they are applied to the CPU. In fact, RPU has the first priority and till there is enough free space on the RPU, the tasks are applied to it. Although this technique is simple and fast, it suffers from inefficiency. For illustration of inefficiency of this method let consider a situation as shown in Fig. 5. As shown, task $T_3$ arrives at time $t_3$ and $T_4$ arrives immediately after $T_3$ (at time $t_4$). According to the *hardware-first* method, $T_3$ is referred to the RPU and occupies large area on it. This causes that the later task ($T_4$) can not be placed on RPU and referred to the CPU, forcedly. As the software execution time of $T_4$ is high, thus the CPU can not execute it before its deadline and it’s rejected, consequently. If HW/SW partitioner partitioned $T_3$ into software, the task $T_4$ would be placed on the RPU and no task would be rejected. Similarly, *software-first* strategy in which the CPU has the first priority in task partitioning, suffers from such inefficiency (drawback).

![Fig. 5. Drawback of hardware-first method in HW/SW partitioning.](image)

This simple example indicates that a good and reasonable method for HW/SW partitioning can improve the performance of the system and reduce the task rejection ratio. To the best of our knowledge, no on-line efficient algorithm has been proposed for HW/SW partitioning in target RC systems.

We base our partitioning method (*area-constrained* partitioning) on the following issues:

1. The more free space is available on the RPU the less the probability of rejection of the future tasks.
2. Large tasks occupy large area on the RPU and reduce the probability of placement for the later tasks. Therefore it is better to apply the large tasks to the CPU as more as possible.

Regarding the above issues, we consider the area of the task as partitioning criterion. In 1D area model, we use task width ($w_i$) instead of task area ($A_i$) as partitioning criterion and compare the width of each task with $W_{FS}^{FS}$. Parameter $W_{FS}^{FS}$ is the width of the maximum existing free space at time $t_i$. This parameter is determined by scanning the total width of
HW/SW Partitioning Algorithm

<table>
<thead>
<tr>
<th>Partitioner($T_i$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: if ($w_i/W_i^{FS}$) &gt; threshold then</td>
</tr>
<tr>
<td>2: Return (SW)</td>
</tr>
<tr>
<td>3: Else</td>
</tr>
<tr>
<td>4: Return (HW)</td>
</tr>
<tr>
<td>5: End if</td>
</tr>
<tr>
<td>6: End</td>
</tr>
</tbody>
</table>

Fig. 6. Pseudo code of HW/SW partitioning algorithm.

RPU at time $t_i$. If the value of $W_i^{FS}$ is zero at a given time it will be set to one. The Pseudo code of the partitioning method has been shown in Fig. 6. The value of threshold in the algorithm is set based on the experiments. Our experiments show that the range of 0.27-0.43 may be a good choice for selecting the value of threshold. The effect of threshold on the quality of results obtained by proposed HW/SW partitioning algorithm has been shown in Fig. 21. As seen in the figure, the range of 0.27-0.43 is an appropriate interval for choosing the value of threshold.

In fact, our partitioning algorithm performs partitioning based on the current situation of RPU. For example, if there is small free space on the RPU, all of the arriving tasks which have large area compared to the available free space will be applied to the CPU, firstly. Of course, if CPU could not schedule them, they will be returned to the RPU, according to our partitioning and scheduling policy.

Partitioning by itself can not guarantee the feasibility of scheduling or placement of the task on RPU or CPU. In fact, the schedulability test of the partitioned task is performed by the HW or SW scheduler. If the partitioned task can not be scheduled by the assigned scheduler, it will be moved to the other scheduler. If the other scheduler can not schedule the task too, it will be rejected.

4.3 On-line SW Scheduler

SW scheduler acts in straightforward method. SW scheduler calculates the earliest start time (EST) of the task, regarding the SW execution time of the previously scheduled tasks which are existing in the SW queue. After EST calculation for the input task, the earliest finish time of the tasks can be easily calculated. Following equations are used to calculate EST and EFT of the task.

$$EST_i = \sum_{T_j \in SQ} e_j^i + (e_j^i + st_k)$$

(4)

$SQ$: {set of tasks which are existing in SW queue}

$$EFT_i = EST_i + e_i^i$$

(5)

Where $T_k$ is the task which is currently executed on CPU and $st_k$ is the execution start time of it.

If $EFT_i \leq d_i$ then the task is scheduled on the CPU and its scheduling time is equal to its EST. but if $EFT_i > d_i$ then the deadline constraint can not be satisfied by CPU and the task is moved to HW scheduler or rejected. Fig. 7 shows the pseudo code of SW scheduler algorithm.
4.4 On-line HW Scheduler

HW tasks which are applied to the HW scheduler should be scheduled and placed on the RPU. In general, the tasks in HW scheduler are divided into three categories at any given time:

(1) Running tasks which are currently placed and are running on the RPU.
(2) Scheduled tasks which have been scheduled on the RPU but their start times are at future time. These tasks are waiting in HW queue.
(3) Arrived tasks which are applied to the HW scheduler and still have not been scheduled.

As stated in section 2.4, we use free space management technique to schedule and place the arrived tasks on the RPU. Fig. 8 shows an example of RPU with two running tasks. A new task ($T_3$) arrives at time $t_3$ and should be placed on the RPU before its deadline ($d_3$). In order to place $T_3$, we find the maximum empty rectangles (MERs) \[5, 24\] in time interval ($t_3, d_3$). In fact, we use the concept of MER for 1D area model. Although in 1D area model only one dimension of RPU (the width of RPU) is considered, we consider the time as the second dimension for defining MER. Figs. 8 (a) and (b) show the examples of MERs. In Fig. 8 (b), there are 4 running tasks and one scheduled task ($T_4$). Each MER has been indicated by its diagonal line.

We denote each MER by $M_i$ and characterize it using a 4-tuple $M_i = (S_{i}^{\text{MER}}, L_{i}^{\text{MER}}, W_{i}^{\text{MER}}, TH_{i})$, where $S_{i}^{\text{MER}}, L_{i}^{\text{MER}}, W_{i}^{\text{MER}}$ and $TH_{i}$ denote the start time of $M_i$, horizon-
tal location of bottom-left corner of $M_i$, width of $M_i$ and time height (time duration) of $M_i$, respectively.

Also, we define the parameter setup delay for each MER at any given time as following that indicates the time delay required to utilize the MER.

$SD_i^{MER} = S_i^{MER} - t_c$  

(6)

Where, $t_c$ is the current time. For example, in Fig. 8 (a), $SD_1^{MER}$, $SD_2^{MER}$ and $SD_3^{MER}$ at time $t_3$ are 5, 3 and 0, respectively.

Several fast and efficient algorithms such as [38, 39] have been proposed to find MERs which can be used. We define feasible MER as following:

**Definition 4** Feasible MER is the MER, $M_j$, which can accommodate the newly arrived task $T_i$. For this purpose, following constraints should be satisfied at arrival time of $T_i$:

**Temporal Constraints:** $(S_i - SD_j^{MER} \geq 0) \land (TH_j^{MER} - e_i^h \geq 0), \quad (7)$

**Spatial Constraint:** $(W_j^{MER} - w_i \geq 0). \quad (8)$

Where $S_i$ is the slack of the arrived task as defined in section 4.1.

An important problem in HW task scheduling and placement is choosing the best MER among the several feasible MERs. Different recent work use first-fit, best-fit [26, 35] or fragmentation aware [36, 38] policy to choose the best MER among several feasible MERs.

For illustrating the drawback of these techniques, let consider an example as shown in Fig. 9. In this example, there are two running tasks on the RPU. Fig. 9 (a) shows all MERs at time $t_5$.

A new task $T_3$ arrives at time $t_3$ to be scheduled and placed on the RPU. Fig. 9 (b) shows all feasible placements for $T_3$. According to first-fit, best-fit or fragmentation-aware methods, $M_1$ is selected for placement. As indicated in the figure, the next arriving task, $T_4$, arrives at time $t_4$. Since $T_4$ has higher urgency than $T_3$ (see the table in the figure), then it can not be placed on the RPU and it is rejected. But if task $T_3$ was placed on $M_1$ or

<table>
<thead>
<tr>
<th>$a_i$</th>
<th>$d_i$</th>
<th>$w_i$</th>
<th>$e_i^h$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1$</td>
<td>2</td>
<td>11</td>
<td>4</td>
</tr>
<tr>
<td>$T_2$</td>
<td>3</td>
<td>9</td>
<td>3</td>
</tr>
</tbody>
</table>

Fig. 9. Drawback of fist-fit and best-fit policy in task placement.
$M_2$, task $T_4$ could be placed on $M_3$ and would not be rejected. In fact, the urgency of the future task has not been considered in placement of current task. Generally, such techniques (first-fit, best-fit and fragmentation-aware) suffer from two main drawbacks:

1. They don’t consider the requirements of future tasks in placement of currently arrived tasks.
2. They focus only on satisfaction of spatial constraints of placement and don’t try to optimize temporal constraints. In the other words, they use only spatial metrics for placement.

Aforementioned example shows that choosing different feasible placements for a task affects the probability of rejection of the later tasks. Also the example shows that choosing the best scheduling time for a newly arrived task strongly depends on the characteristics of the next arriving tasks (future tasks).

To the best of our knowledge, so far, none of the proposed on-line scheduling algorithms has considered the influence of future tasks on the scheduling and placement of currently arrived task. In the rest of this paper, we propose an on-line scheduling algorithm in which the characteristics of the future tasks are predicted and used to make correct decision for current task placement. In the other words, the placement of currently arrived task is carried out in such a way that there is the most matching between future task requirements and future MERs (the MERs which are formed after placement of current task). Also, we take temporal constraints in addition to spatial constraints into account for placement of the tasks.

In what follows, we explain the method of predicting future task requirements and then describe the method of selecting the best MER for placement.

### 4.4.1 Future task prediction

We use poison probability distribution for estimating the probability of arriving tasks in the future. Thus the probability of arriving $k$ tasks in next time interval $\Delta t$ is defined as following.

$$P_{\Delta t}(k) = \frac{\lambda^k \times e^{-\lambda}}{k!}.$$  \hspace{1cm} (9)

Where, $\lambda$ is the average number of arrived tasks in past time intervals ($\Delta t$).

Regarding the above equation, we can calculate the next time interval in which the probability of arriving one task, $P_{\Delta t}(1)$, is the maximum. Following equations show that $\lambda = 1$ leads to the maximum value for $P_{\Delta t}(1)$.

$$P_{\Delta t}(1) = \lambda e^{-\lambda}$$  \hspace{1cm} (10)

$$\frac{d(P_{\Delta t}(1))}{d\lambda} = (1-\lambda)e^{-\lambda} = 0 \Rightarrow \lambda = 1$$  \hspace{1cm} (11)

$$\lambda = 1 \Rightarrow \frac{n}{P} \times \Delta t = 1 \Rightarrow \Delta t = \frac{P}{n}$$  \hspace{1cm} (12)
Where \( n \) is the number of arrived tasks in past time window. We use a time window \((t_c - P, t_c)\) with length \( P \) to determine the average number of arrived tasks in each time unit in the past. As stated before, \( t_c \) denotes the current time.

Regarding above equations, in the next \( \Delta t \), the probability of arriving one task is the maximum. We use \( S, e^h, w \) and \( T_I \) as slack, HW execution time, width and task interval of the next arriving task (future task). These parameters are determined as following,

\[
\begin{align*}
S &= \text{average slack of the arrived tasks during past time window} \\
\overline{e}^h &= \text{average HW execution time of the arrived tasks during past time window} \\
w &= \text{average width of the arrived tasks during past time window} \\
\overline{T_I} &= \text{average task interval of the arrived tasks during past time window}
\end{align*}
\]

Above parameters are calculated at each task arrival time by the parameter estimation part of the algorithm described in section 4.1. We use these parameters, to select the best feasible MER for placing the current task.

### 4.4.2 Selecting the best MER for task placement

Placing the new task in each of feasible MERs leads to formation of new MERs in the next \( \Delta t \), so called future MERs and denoted by \( MER' \). Our proposed algorithm tries to maximize the matching between future MERs (\( MER' \)) and future task requirements in the next \( \Delta t \). The proposed algorithm includes the following steps:

- Each feasible MER (\( M_f \)) is selected and the currently arrived task is placed in \( M_f \), tentatively. Afterward the next steps are performed. Current time (\( t_c \)) is moved forward to \((t_c + \Delta t)\) and all events (task terminations and task starts) are simulated.

All newly formed MERs are determined in time interval \((t_c + \Delta t, t_c + \Delta t + \overline{T_I})\). We denote them as \( MER' \) set.

1. The matching function is calculated for the new situation as following,

\[
MF^{'MER} = \sum_{M_j \in MER'} \left( \frac{(\overline{S} - SD^{'MER})}{\text{Max}(\overline{S}, SD^{'MER})} + \frac{(TH_k^{'MER} - \overline{e}^h)}{\text{Max}(TH_k^{'MER}, e^h)} + \frac{(W_k^{'MER} - \overline{w})}{\text{Max}(W_k^{'MER}, \overline{w})} \right)
\]

In fact, matching function indicates the matching value between characteristics of future MERs (\( MER' \)) and requirements of future tasks. The higher matching value is the less probability of rejection of next arriving tasks and the less the probability of fragmentation in the future.

2. Steps 1 to 4 are repeated for the other feasible MERs and matching function is calculated for each MER.

3. The MER which has the highest matching value is selected for placing the current task.

Large values of matching function indicate that the size of future feasible MERs are large compared to the size of future tasks. This decreases the probability of fragmentation in the future. In fact, matching function takes the temporal and spatial metrics into account for placement of the tasks.
### HW Scheduling Algorithm

<table>
<thead>
<tr>
<th>HW Scheduler(Ti)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: MER_set = MER_determine(a, d)</td>
</tr>
<tr>
<td>2: Requirement = Future_task_requirements()</td>
</tr>
<tr>
<td>3: Feasible_set = Feasible_determin(MER_set)</td>
</tr>
<tr>
<td>4: If Feasible_set = Ø then</td>
</tr>
<tr>
<td>5: Return (false, Ø)</td>
</tr>
<tr>
<td>6: Exit</td>
</tr>
<tr>
<td>7: End if</td>
</tr>
<tr>
<td>8: For all Mj ∈ {Feasible_set} Do</td>
</tr>
<tr>
<td>9: MER′_set = MER′_determine(Mj, Ti)</td>
</tr>
<tr>
<td>10: MFj = Matching_function(MER′_set, Requirement)</td>
</tr>
<tr>
<td>11: End for</td>
</tr>
<tr>
<td>12: M ← Select Mj with the highest MFj</td>
</tr>
<tr>
<td>13: Move Ti into HW_queue</td>
</tr>
<tr>
<td>14: Return (true, scheduling_parameters)</td>
</tr>
<tr>
<td>15: End</td>
</tr>
</tbody>
</table>

Fig. 10. Pseudo code of HW scheduling algorithm.

Fig. 10 shows the pseudo code of on-line HW scheduling algorithm. Function MER_determine(a, d) finds all MERs in time interval (a, d). Function MER′_determine (Mj, Ti) finds all MER′ after placing Ti in Mj. Function future-task-requirements() estimates the characteristics of the future task and function Feasible_determin(MER_set) determines the feasible MERs among all existing MERs according to Definition 4.

#### 4.5 Overall Partitioning and Co-scheduling Algorithm

Fig. 11 shows the pseudo code of overall HW/SW partitioning and co-scheduling algorithm. As seen in the figure, function Param_estimate(T) receives the input task set, T, and calculates the required parameters such as $\overline{S}$, $\overline{w}$, $\overline{e^d}$ and so on which are used in HW scheduler. Function Prioritizer(T) performs urgency-based prioritizing of the input tasks according to their urgencies.

After prioritizing, the task with the highest priority is selected and partitioned into HW or SW by partitioner(Ti) function. If the task is partitioned into SW it will be applied to the CPU for scheduling. Function SW_scheduler(Ti) tries to schedule the SW task and it will return true in addition to the scheduling parameters, if it can schedule the task on CPU. Otherwise, false is returned and the task is moved to HW scheduler. Function HW_scheduler(Ti) tries to schedule and place the returned task on the RPU as explained in section 4.4. If it succeed, it will return true in addition to the scheduling parameters. Otherwise, false is returned and the task is rejected. Similar procedure is carried out for the tasks which are partitioned into HW by partitioner.

#### 5. COMPLEXITY ANALYSIS OF ALGORITHM

The overall partitioning and co-scheduling algorithm includes four main steps: task prioritizing, HW/SW partitioning, SW scheduling and HW scheduling. Among these steps, only HW scheduling has considerable run-time complexity. HW scheduling algorithm, by
On-line Partitioning and Co-Scheduling Algorithm

```
Co_Scheduler(T)
1: Parameters = Param_estimate(T)
2: Prioritizer(T)
3: While T ≠ Ø Do
4:   T_i ← Select T_i with the highest priority
5:   If partitioner(T_i) = SW then
6:     (r, scheduling_parameters) = SW_scheduler(T_i)
7:     If r = false then
8:       (r, scheduling_parameters) = HW_scheduler(T_i)
9:       If r = false then
10:          Reject (T_i)
11:          Exit
12:     Else
13:        Return (true, scheduling_parameters)
14:     End if
15:   Else
16:     (r, scheduling_parameters) = HW_scheduler(T_i)
17:     If r = false then
18:       (r, scheduling_parameters) = SW_scheduler(T_i)
19:       If r = false then
20:          Reject (T_i)
21:          Exit
22:     Else
23:        Return (true, scheduling_parameters)
24:     End if
25:   End if
26:   Remove T_i from set T
27: End while
```

Fig. 11. Pseudo code of overall HW/SW partitioning and co-scheduling algorithm.

itself, includes three steps: finding feasible MERs, estimation of future task parameters, calculating the matching function for each feasible MER. Run time complexity of the algorithm strongly depends on the number of feasible MERs and the number of future MERs which are formed after tentative placements of the current tasks. The complexity of finding feasible MERs in the worst case is $O(W \times H)$, where $W$ is the number of columns in RPU and $H$ is the maximum task interval (in terms of time units) in which the MERs are searched. The complexity of calculating matching function for each feasible MER in the worst case is $O(W \times H)$. If the number of feasible MERs at task arrival time is $N$, the complexity of overall algorithm will be $O(N \times W \times H)$. In the worst case, $N$ is the sum of number of running tasks and number of scheduled tasks on the RPU. Our experimental simulations show that the algorithm running time on a 1 GHz Pentium III computer is less than 0.25 ms. Although the computation power of current embedded processors are small com-
pared to that of Pentium III, we believe this time complexity is acceptable in comparison with reconfiguration time of the RPUs.

6. EXPERIMENTAL EVALUATION

6.1 Simulation Setup

We have constructed a discrete time simulation framework in C++ language to experimentally evaluate the performance of the proposed algorithm. Run-time measurements have been conducted on a 1 GHz Pentium III computer. The simulated device consists of $96 \times 64 = 6144$ reconfigurable units (RCUs), which corresponds to xilinx’s XCV-1000 FPGA.

Many different CPUs, from low performance embedded processors to high performance processors, can be used in target RC system model. The main assumption for CPU is that the CPU can run only one task at a time. The performance of the CPU, only affects the SW execution time of the tasks. In our simulation setup, execution time ratio of the tasks ($e_i^s/e_i^h$) is considered in range 1.2-2.5 (see Table 1). Since, the HW execution time of the tasks has been considered in range 5-100 time units (see Table 1); therefore, SW execution time of the tasks is in range 6-250 time units. All kinds of CPU which provide this performance can be used in our model (such as ARM Cortex processors).

<table>
<thead>
<tr>
<th>Task Group</th>
<th>Width ($w_i$)</th>
<th>HW execution time ($e_i^h$)</th>
<th>Slack ($d_i - a_i - e_i^h$)</th>
<th>Execution time ratio ($e_i^s/e_i^h$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>[2, 15]</td>
<td>[30, 45]</td>
<td>[5, 30]</td>
<td>1.2-2.5</td>
</tr>
<tr>
<td>B</td>
<td>Uniform distribution in interval [2, 45]</td>
<td>Uniform distribution in interval [5, 100]</td>
<td>Uniform distribution in interval [1, 100]</td>
<td>Uniform distribution in interval [1, 100]</td>
</tr>
</tbody>
</table>

A. 25% 50% 25% 25% 50% 25% 25% 50% 25% Uniform distribution

B. Uniform distribution in interval [2, 45]

We have conducted our experiments on three groups of input tasks as following,

1. Task groups A and B: synthetic tasks,
2. Task group C: real tasks.

In task groups A and B, we use task sets with randomly generated parameters. The probability distribution of task parameters is based on real tasks in real world applications. Taking the sizes and execution times of the common cores (tasks) mapped to RPU into account, we have simulated the synthetic tasks. The parameters of the tasks in different groups have been distributed as Table 1. The probability distribution of task parameters is general enough to show the effectiveness of the algorithm.

Fifty task sets have been generated based on the synthetic tasks in each task group and the simulation results have been averaged. In each task set, the number of tasks has been set to 50. The number of tasks has been set to achieve the confidence level of 95% with an error range of at most $\pm 6\%$. 
For each task in a task set, we generate a random arrival time. Hence two or more tasks in a task set may have the same arrival time ($a_i$) which means that they arrive simultaneously.

We use a workload parameter instead of the arrival time distribution in the following figures. We denote the set of tasks arriving in the time interval $\Delta T$ with $T$. $\Delta T$ is set to span from the arrival of the first task to the arrival of the last task plus its execution time. Every task $T_i \in T$ occupies $w_i \times h_i$ reconfigurable units on the device area but as our area model is 1D, we use only $w_i$ in workload definition. In Eq. (14), $W$ is the total width of the RPU.

$$Workload = WL_{T,\Delta T} = \frac{\sum_{T_i \in T} w_i \times e_i^h}{W \times \Delta T}$$

(14)

In task group C, we used real tasks of JPEG and MPEG encoding/decoding jobs. We have partitioned JPEG/MPEG encoding/decoding jobs into 13 tasks according to their task graph. These tasks include: RGB2YCrCb, motion estimator, motion compensator, quantizer, dequantizer, DCT, IDCT, huffman encoder/decoder, run length encoder/decoder, entropic coder/decoder.

Two or several different implementations of each aforementioned task have been included in task group C. Totally, task group C consists of 35 real tasks. These tasks are the most common tasks which are used in image and video processing. Twenty task sets have been generated using these tasks and the simulation results have been averaged. We have used workload parameter instead of task arrival time in figures. The experiments have been carried out for different workloads.

6.2 Simulation Results

We have compared the results of simulations for four different algorithms: our proposed future-aware scheduling (FA) algorithm, window-based stuffing (WBS) algorithm [22] which is the improved version of stuffing algorithm [26], fragmentation-aware (FR-H) algorithm presented by handa [38] and fragmentation-aware (FR-C) algorithm presented by cui [36]. For all algorithms, we have used the same prioritizing method and HW/SW partitioning method as described in sections 4.1 and 4.2 respectively. In fact, only the scheduling quality of the algorithms has been compared. Figs. 12-14 show the values of TRR against the workload for different scheduling algorithms and different task groups.
Figs. 15-17 indicate the results of experiments for two different prioritizing methods used in our proposed scheduling algorithm. As seen, the results show that our proposed urgency-based prioritizing outperforms the EDF-based prioritizing method.

Figs. 18-20 show the results of experiments for using three different HW/SW partitioning policies in the proposed algorithm. HW-first, SW-first and area-constrained poli-
cies have been used as three different partitioning policies. Obtained results show the superiority of area-constrained policy.

Fig. 21 indicates the effect of parameter threshold on result quality of the proposed HW/SW partitioning algorithm. We pointed out this figure in section 4.2.

In Fig. 22, the runtime of the different algorithms have been compared. Runtime measurements have been conducted on a 1 GHz Pentium III computer. In runtime measurement of the algorithms, one time unit corresponds to 50ms, therefore according to Table 1, HW execution time of the tasks is in the range 250ms-5000ms and the slack of the tasks is in the range 50ms-5000ms. Regarding above parameters in the measurement, the runtime overhead of the proposed algorithm seems to be acceptable compared to HW execution time and slack of the tasks.

7. CONCLUSION

In this work, we described the problem of on-line partitioning and co-scheduling for RC systems with real-time, non-preemptive task model and 1D area model. We reviewed the related work. Afterward, we presented our proposed algorithm which performs on-line integrated partitioning and co-scheduling based on maximal matching between requirements of future tasks and the characteristics of future MERs. Also, we proposed urgency-based method instead of EDF-based method for input tasks prioritizing. We proposed
area-constraint policy for HW/SW partitioning, too. The complexity of proposed algorithm was estimated that shows acceptable overhead. Simulation results show considerable improvement in TRR for randomly generated task sets and real task sets in different workloads.

REFERENCES


Maisam M. 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. Now he is Ph.D. student in electrical engineering in Iran University of Science and Technology. 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.

Hadi Shahriar Shahhoseini received B.S. degree in Electrical Engineering from University of Tehran, Tehran, Iran, in 1990, M.S. degree from Azad University of Tehran in 1994 and Ph.D. degree in Electrical Engineering from Iran University of Science and Technology, Tehran, Iran in 1999. He is an Assistant 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 100 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 co-ordinator in Middle-East Countries.