Computers and Electrical Engineering xxx (2013) xxx–xxx

Contents lists available at SciVerse ScienceDirect

Computers and Electrical Engineering

journal homepage: www.elsevier.com/locate/compeleceng

Online scheduling and placement of hardware tasks with multiple variants on dynamically reconfigurable field-programmable gate arrays ☆

Thomas Marconi *

School of Computing, National University of Singapore, Computing 1, 13 Computing Drive, Singapore 117417, Singapore
Computer Engineering Lab., TU Delft., Building 36, Room: HB10.320, Mekelweg 4, 2628 CD Delft, The Netherlands

ARTICLE INFO

Article history:
Available online xxxx

ABSTRACT

Hardware task scheduling and placement at runtime plays a crucial role in achieving better system performance by exploring dynamically reconfigurable Field-Programmable Gate Arrays (FPGAs). Although a number of online algorithms have been proposed in the literature, no strategy has been engaged in efficient usage of reconfigurable resources by orchestrating multiple hardware versions of tasks. By exploring this flexibility, on one hand, the algorithms can be potentially stronger in performance; however, on the other hand, they can suffer much more runtime overhead in selecting dynamically the best suitable variant on-the-fly based on its runtime conditions imposed by its runtime constraints. In this work, we propose a fast efficient online task scheduling and placement algorithm by incorporating multiple selectable hardware implementations for each hardware request; the selections reflect trade-offs between the required reconfigurable resources and the task runtime performance. Experimental studies conclusively reveal the superiority of the proposed algorithm in terms of not only scheduling and placement quality but also faster runtime decisions over rigid approaches.

1. Introduction

Nowadays, digital electronic systems are used in a growing number of real life applications. The most flexible and straight forward way to implement such a system is to use a processor that is programmable and can execute wide variety of applications. The hardware, a general-purpose processor (GPP) in this case, is usually fixed/hardwired, whereas the software ensures the computing system flexibility. Since such processors perform all workloads using the same fixed hardware, it is too complex to make the hardware design efficient for a wide range of applications. As a result, this approach cannot guarantee the best computational performance for all intended workloads. Designing hardware devices for a particular single application, referred as Application-Specific Integrated Circuits (ASICs), provides a system with the most efficient implementation for the given task, e.g., in terms of performance but often area and/or power. Since this requires time consuming and very costly design process along with expensive manufacturing processes, it is typically not feasible in both: economic costs and time-to-market. This solution, however, can become interesting when very high production volumes are targeted. Another option that allows highly flexible as well as relatively high performance computing systems is using reconfigurable devices,

☆ Reviews processed and approved for publication by Editor-in-Chief Dr. Manu Malek.

* Addresses: School of Computing, National University of Singapore, Computing 1, 13 Computing Drive, Singapore 117417, Singapore, Computer Engineering Lab. TU Delft, Building 36, Room HB10.320, Mekelweg 4, 2628 CD Delft, the Netherlands. Tel.: +65 6516 2727/+31 15 2786196; fax: +65 6779 4580/+31 15 2784898.

E-mail address: thomas_marconi@yahoo.com

0045-7906/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.compeleceng.2013.07.004

such as Field-Programmable Gate Arrays (FPGAs). This approach aims at the design space between the full custom ASIC solution and the general-purpose processors. Often platforms of this type integrate reconfigurable fabric with general-purpose processors and sometimes dedicated hardwired blocks. Since such platforms can be used to build arbitrary hardware by changing the hardware configuration, they provide a flexible and at the same time relatively high performance solution by exploiting the inherent parallelism in hardware. Such systems where hardware can be changed at runtime are referred as runtime reconfigurable systems. A system involving partially reconfigurable devices can change some parts during runtime without interrupting the overall system operation [1,2].

Recently, dynamically reconfigurable FPGAs have been used in some interesting application areas, e.g., data mining [3], automobile [4], stencil computation [5], financial computation [6], Networks on Chip (NoC) [7], robotics [8], Software Defined Radio (SDR) [9], among many others. Exploiting partially reconfigurable devices for runtime reconfigurable systems can offer reduction in hardware area, power consumption, economic cost, bitstream size, and reconfiguration time in addition to performance improvements due to better resource customization (e.g. [1,2]). To make better use of these benefits, one important problem that needs to be addressed is hardware task scheduling and placement. Since hardware on-demand tasks are swapped in and out dynamically at runtime, the reconfigurable fabric can be fragmented. This can lead to undesirable situations where tasks cannot be allocated even if there would be sufficient free area available. As a result, the overall system performance will be penalized. Therefore, efficient hardware task scheduling and placement algorithms are very important.

Hardware task scheduling and placement algorithms can be divided into two main classes: *offline* and *online*. Offline assumes that all task properties (e.g., arrival times, task sizes, execution times, and reconfiguration times) are known in advance. The offline version can then perform various optimizations before the system is started. In general, the offline version has much longer time to optimize system performance compared to the online version. However, the offline approach is not applicable for systems where arriving tasks properties are unknown beforehand. In such general-purpose systems, even processing in a nondetermistic fashion of multitasking applications, the online version is the only possible alternative. In contrast to the offline approach, the online version needs to take decisions at runtime; as a result, the algorithm execution time contributes to the overall application latency. Therefore, the goal of an online scheduling and placement algorithm is not only to produce better scheduling and placement quality, they also have to minimize the runtime overhead. The online algorithms have to quickly find suitable hardware resources on the partially reconfigurable device for the arriving hardware tasks. In cases when there are no available resources for allocating the hardware task at its arrival time, the algorithms have to schedule the task for future execution.

Although a number of online hardware task scheduling and placement algorithms on partially reconfigurable devices have been proposed in the literature, no study has been done aimed for dynamically reconfigurable FPGA with multiple selectable hardware versions of tasks, paving the way for a more comprehensive exploration of its flexibility. Each hardware implementation for runtime systems with rigid algorithms, considering only one hardware implementation for handling each incoming request, is usually designed to perform well for the heaviest workload. However, in practice, the workload is not always at its peak. As a result, the best hardware implementation in terms of performance tends to use more resources (e.g. reconfigurable hardwares, electrical power, etc) than what is needed to meet its runtime constraints during its operation. Implementing a hardware task on FPGA can involve many trade-offs (e.g. required area, throughput, clock frequency, power consumption, energy consumption). These trade-offs can be done by controlling the degree of parallelism [10–15], using different architecture [16,17], and using different algorithms [18,19]. Given this flexibility, we can have multiple hardware implementations for executing a task on FPGA. To better fit tasks to reconfigurable device, since partially reconfiguration can be performed at runtime, we can select dynamically the best variant among all possible implemented hardwares on-the-fly based on its runtime conditions (e.g. availability of reconfigurable resources, power budget, etc) to meet its runtime requirements (e.g. deadline constraint, system response, etc) at runtime. Therefore, it is worthwhile to investigate and build algorithms for exploring this flexibility more, creating the nexus between the scheduling and placement on dynamically reconfigurable FPGAs and the trade-offs of implementing hardware tasks. In this work, we focus on online scheduling and placement since we strongly believe it represents a more generic situation, supporting execution of multiple applications concurrently. In a nutshell, we propose to incorporate multiple hardware versions for online hardware task scheduling and placement on dynamically reconfigurable FPGA for benefiting its flexibility in gaining better runtime performance with lower runtime overhead.

In this article, first of all, we introduce an idea to take into account multiple hardware versions of hardware tasks during scheduling and placement at runtime on dynamically reconfigurable FPGAs. On one hand, because of its variety of choices, in this case, the algorithms have more flexibility in choosing the right version for running each arriving task, gaining a better performance. On the other hand, the algorithms can suffer high runtime overhead since they not only need to find the right location and timing for running the incoming tasks but also they need to choose the best implementation among multiple hardware variants. Next, we do comprehensive computer simulations to show that the idea of multiple implementations of tasks significantly improves the performance of reconfigurable computing systems; however, it suffers high runtime overhead. Since we target a realistic online scenario where the properties of tasks are unknown at compile time, time taken by the algorithms are considered as an overhead to the application overall execution time; hence this issue needs to be solved properly. To handle this issue, we subsequently introduce a technique to amortize this high runtime overhead by reducing search space at runtime, referred as *pruning capability*. With this technique, the algorithm quickly determines all reconfigurable units that cannot be assigned to hardware tasks, avoiding exhaustively searching the space yet yielding the same runtime performance. We then apply this technique to build an efficient algorithm to tackle the scheduling and placement

Please cite this article in press as: Marconi T. Online scheduling and placement of hardware tasks with multiple variants on dynamically reconfigurable field-programmable gate arrays. Comput Electr Eng (2013), http://dx.doi.org/10.1016/j.compeleceng.2013.07.004

problem with multiple hardware versions without suffering high runtime overhead, named as *pruning moldable (PM)*. Finally, we evaluate the proposed algorithm by computer simulations using both synthetic and real hardware tasks.

The main contributions of this work are:

1. An idea to incorporate multiple hardware versions of hardware tasks during task scheduling and placement at runtime targeting partially reconfigurable devices, having more flexibility in choosing the right version for running each arriving task in gaining a better performance.
2. Formal problem definition of online task scheduling and placement with multiple hardware versions for reconfigurable computing systems.
3. An idea of FPGA technology independent environment by decoupling area model from the fine-grain details of the physical FPGA fabric, enabling different FPGA vendors providing their consumers with full access to partial reconfigurable resources without exposing all of the proprietary details of the underlying bitstream formats.
4. A technique for shortening runtime overhead by reducing search space drastically at runtime using a notion of *pruning technique*.
5. A fast and efficient algorithm to tackle the scheduling and placement problem for dynamically reconfigurable FPGAs with multiple hardware implementations, named as *pruning moldable (PM)*.
6. An extensive evaluation using both synthetic and real hardware tasks, revealing the superiority of the proposed algorithm in performing hardware task scheduling and placement with much faster runtime decisions.

The rest of the article is organized as follows. A survey of related work is discussed in Section 2. In Section 3, we introduce the problem of online task scheduling and placement with multiple hardware versions targeting dynamically reconfigurable FPGAs. The proposed system model with FPGA technology independent environment is shown in Section 4. In Section 5, our proposed algorithm is presented. The algorithm is evaluated in Section 6. Finally, we conclude in Section 7.

2. Related work

The *Horizon* and *Stuffing* schemes for both one-dimensional (1D) and two-dimensional (2D) area models targeting 2D FPGAs have been proposed in [20]. The Horizon scheme in scheduling hardware tasks only considers each of free regions that has an infinite free time. By simulating future task initiations and terminations, the Stuffing scheme has an ability to schedule arriving tasks to arbitrary free regions that will exist in the future, including regions with finite free time. Simulation experiments in [20] show that the Stuffing scheme outperforms the Horizon scheme in scheduling and placement quality because of its ability in recognizing the regions with finite free time.

In [21], 1D *Classified Stuffing* algorithm targeting 2D FPGAs is proposed to tackle the drawback of the 1D Stuffing algorithm. Classifying the incoming tasks before scheduling and placement is the main idea of the 1D Classified Stuffing algorithm. Experimental results in [21] show that 1D Classified algorithm has a better performance than the original 1D Stuffing algorithm.

To overcome the limitations of both the 1D Stuffing and Classified Stuffing algorithms, 1D *Intelligent Stuffing* algorithm targeting 2D FPGAs is proposed in [22,23]. The main difference of this algorithm compared to the previous 1D algorithms is the additional alignment status of each free region and its handling. The alignment status is a boolean variable steering the placement position of the task within the corresponding free region. By utilizing this addition, as reported in [22,23], the 1D Intelligent Stuffing scheme outperforms the previously mentioned 1D schemes (i.e. Stuffing [20] and Classified Stuffing [21] schemes).

A strategy called as 1D *reuse and partial reuse* (RPR) targeting 2D FPGAs is proposed in [24]. To reduce reconfiguration time, the strategy reuses already placed tasks. As a result, presented in [24], the RPR outperforms the 1D Stuffing algorithm. The extension of algorithm, referred as the Communication-aware, to take into account the data dependency and communications both among hardware tasks and external peripherals is presented in [25].

To tackle the drawback of 2D Stuffing algorithm, a new algorithm targeting 2D FPGAs called as 2D *Window-based Stuffing* is proposed in [26]. By using time windows instead of the time events, the 2D Window-based Stuffing algorithm outperforms previous 2D Stuffing algorithm. The main drawback of the algorithm is its high runtime overhead that renders it unattractive.

In [27], *Compact Reservation* algorithm targeting 2D FPGAs is proposed in an attempt to reduce the excessive runtime overhead of Window-based Stuffing. The key idea of the algorithm is the computation of an earliest available time matrix for every incoming task. The matrix contains the earliest starting times for scheduling and placing the arriving task. The Compact Reservation (CR) algorithm outperforms the original 2D Stuffing algorithm and the 2D Window-based Stuffing algorithm as reported in [27].

All of the above discussed algorithms tend to allocate tasks that potentially block future arriving tasks to be scheduled earlier denoted as "blocking-effect". The first blocking-aware algorithm targeting 2D FPGAs, denoted as *three-dimensional (3D) Compaction* (3DC), is proposed in [28,23] to avoid the "blocking-effect". To empower the algorithm with blocking-awareness, a *3D total contiguous surfaces* (3DTCS) heuristic is introduced. Because of its blocking-awareness, it leads to improved performance over the CR algorithm as shown in [28,23].

Please cite this article in press as: Marconi T. Online scheduling and placement of hardware tasks with multiple variants on dynamically reconfigurable field-programmable gate arrays. Comput Electr Eng (2013), http://dx.doi.org/10.1016/j.compeleceng.2013.07.004

Although all existing algorithms discussed here have been proposed in their efforts to solve problems involving in scheduling and placement on dynamically reconfigurable FPGAs; however, no study has been reported for online hardware task scheduling and placement incorporating multiple hardware versions.

3. Problem definition

A 2D dynamically reconfigurable FPGA, denoted by FPGA (W,H) as illustrated in Fig. 1, contains $W \times H$ reconfigurable hardware units arranged in a 2D array, where each element of the array can be connected with other element(s) using the FPGA interconnection network. The reconfigurable unit located in i^{th} position in the first coordinate (x) and j^{th} position in the second coordinate (y) is identified by coordinate (i,j), counted from the lower-leftmost coordinate $(1,1)$, where $1 \leqslant i \leqslant W$ and $1 \leqslant j \leqslant H$.

A hardware task T_i with n multiple hardware versions, denoted by $T_i = \{T_{i,1}, T_{i,2}, \ldots, T_{i,n}\}$, arrives to the system at arrival time a_i with deadline d_i as shown in Fig. 1. Each version requires a region of size $w \times h$ in the 2D FPGA (W,H) during its lifetime lt; it can be represented as a 3D box in Fig. 1(c) or (d). We call this as a *3D representation of the hardware task*. We define $lt = rt + et$ where rt is the reconfiguration time and et is the execution time of the task. w and h are the task width and height where $1 \leqslant w \leqslant W$, $1 \leqslant h \leqslant H$.

Online task scheduling and placement algorithms targeting 2D FPGAs with multiple selectable hardware versions have to choose the best hardware implementation on-the-fly at runtime among those available hardware variants and find a region of hardware resources inside the FPGA for running the chosen hardware version for each arriving task. When there are no available resources for allocating the hardware task at its arrival time a, the algorithm has to schedule the task for future execution. In this case, the algorithm needs to find the starting time t_s and the free region (with *lower-leftmost corner* (x_1,y_1) and *upper-rightmost corner* (x_2,y_2)) for executing the task in the future. The running or scheduled task is denoted as $T(x_1,y_1,x_2,y_2,t_s,t_f)$, where $t_f = t_s + lt$ is the finishing time of the task. The hardware task meets its deadline if $t_f \leqslant d$. We also call the *lower-leftmost corner* of a task as the *origin* of that task.

The goals of any scheduling and placement algorithm are to minimize the total number of hardware tasks that miss their deadlines and to keep the runtime overhead low by minimizing algorithm execution time. We express the *deadline miss ratio* as the ratio between the total number of hardware tasks that miss their deadlines and the total number of hardware tasks arriving to the system. The *algorithm execution time* is defined as the time needed to schedule and place the arriving task.

4. System model

In this article, each hardware task with multiple hardware versions can arrive at any time and its properties are unknown to the system beforehand. This models real situations in which the time when the user requests system resources for his/her usage is unknown. As a result, the envisioned system has to provide support for runtime hardware tasks placement and scheduling since this cannot be done statically at system design time. To create multiple hardware variants, it can be done

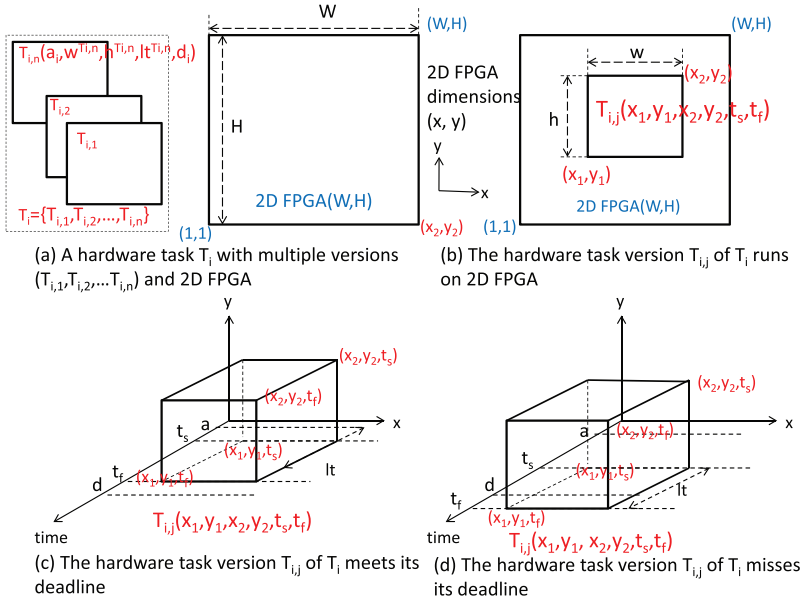

Fig. 1. Problem of online task scheduling and placement on 2D partially reconfigurable FPGAs with multiple hardware versions.

Please cite this article in press as: Marconi T. Online scheduling and placement of hardware tasks with multiple variants on dynamically reconfigurable field-programmable gate arrays. Comput Electr Eng (2013), http://dx.doi.org/10.1016/j.compeleceng.2013.07.004

by controlling the degree of parallelism [10–15], implementing hardwares using different architectures [16,17] and algorithms [18,19]. Some of known techniques to control the degree of parallelism are loop unrolling with different unrolling factors [10–13], recursive variable expansion (RVE) [14], and realizing hardwares with different pipeline stages [15].

Similar to other work, we assume that each hardware version requires reconfigurable area with rectangular shape and can be placed at any location on the reconfigurable device. Our task model includes all required reconfigurable units and interconnect resources. Each hardware version is first designed using a hardware description language and after that is placed and routed by commercial FPGA synthesis Computer-Aided Design (CAD) tools to obtain functionally equivalent modules that can replace their respective software versions. At this step of the design process we can use the synthesis results to extract the task sizes for the used FPGA fabric. The output of the synthesis is the configuration bitstream file. The reconfiguration time of the targeted technology is the elapsed time needed to load the bitstream file to the device using its integrated configuration infrastructure. The two key ingredients are the configuration data size (the bitstream length in number of bits) and the throughput of the internal FPGA configuration circuit. As an example, the *Internal Configuration Access Port* (ICAP) [29] of Virtex 4 FPGAs from Xilinx can transport 3200 million bits/second and will load a bitstream of size 51 Mbits in 15.9 ms. The last parameter, the task execution time is specified by the time needed to process a unit of data (referred as: *Execution Time Per Unit of Data*, ETPUD) and the overall data size to be processed (i.e. how much data need to be processed). Please note that for some applications, the task execution time is also dependent on the exact data content (e.g., as in the case of Viterbi and *Context-Adaptive Binary Arithmetic Coding* (CABAC)). In such applications, even when processing the same amount of data, the elapsed time will be different when the input data content changes. To address data dependent task execution times, we envision two solutions: *worst case execution time* scenario and *notification on task completion* hardware support.

In this article, we assume the worst case execution time scenario in which we use the task execution time when processing the worst case input data content. In such scenario, it is possible that the actual task completion can happen earlier than the scheduled completion time resulting in idle times that can not be used by the algorithm. In addition, such tasks will cause additional wasted area that cannot be utilized immediately by other hardware tasks. In such non-optimal scenario, however, the overall computing system will operate correctly. Please note that the chosen scenario is the worst case in respect to the proposed placement and scheduling algorithm due to the introduced overheads in task execution time and wasted area. The second solution requires dedicated hardware support for feedback signaling when the running tasks complete, however, as mentioned earlier this can additionally improve the overall system performance. Some existing systems already have the necessary ingredients required to implement such support. For example, in the Molen architecture [30], the sequencer is aware of the hardware task start and completion timing. The only necessary extension in this case is to provide a way to pass this information to the scheduling and placement algorithm and make it aware of running tasks completion. With this knowledge, the algorithm can make data content dependent tasks scheduling more efficient. This approach is outside of the scope of this work without loosing generality of our proposal. Even more the selected worst case execution scenario is less beneficial for the scheduling and placement algorithm presented in this study. We are strongly convinced that both types of systems will be able to benefit from the proposed algorithms.

The assumed overall system model used in our study is sketched in Fig. 2 consisting of two main devices: the general-purpose processor (GPP) and the reconfigurable device (e.g. FPGA). All hardware task bitstream images are available in a repository resuming in the main memory (not explicitly shown on the figure) and can be requested by any running application on the GPP by using a dedicated operating system (OS) call. In response to such request, the OS will invoke the scheduling and placement algorithm (S&P) to find the best location on the FPGA fabric for the requested hardware task. Once appropriate location is found, the Translator will resolve the coordinates by transforming the internal, technology independent model representation to the corresponding low level commands specific for the used FPGA device. The Loader reads the

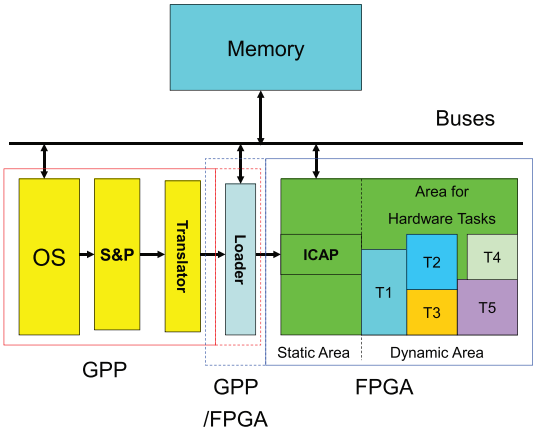

Fig. 2. System model (OS: operating system, S&P: scheduling and placement algorithm).

Please cite this article in press as: Marconi T. Online scheduling and placement of hardware tasks with multiple variants on dynamically reconfigurable field-programmable gate arrays. Comput Electr Eng (2013), http://dx.doi.org/10.1016/j.compeleceng.2013.07.004

task configuration bitstream from the repository and sends it to the internal configuration circuit, e.g., ICAP in case of Xilinx, to partially reconfigure the FPGA at the specific physical location provided by the Translator. After reconfiguring the FPGA fabric the requested hardware task execution is started immediately to avoid idle hardware units on the reconfigurable fabric. When the hardware area is fully occupied, the scheduling and placement algorithm (S& P) schedules the task for future execution at predicted free area places at specific locations and specific starting times. The Loader can be implemented at either the GPP as OS extension or in the Static Area of the FPGA. If the Loader is implemented in the GPP, the communication between the Loader to the ICAP is performed using the available off-chip connection. For implementations in the FPGA, the Loader connects to the ICAP internally. Similar to [31], dedicated buses are used for the interconnect on chip. Those buses are positioned at every row of the reconfigurable regions to allow data connections between tasks (or tasks and Inputs/Outputs(I/Os)) regardless of the particular task sizes.

To illustrate the above processes at different stages of the system design and normal operation, we will use a simple example of hardware task creation as depicted in Fig. 3. The hardware task in our example is a simple finite impulse response (FIR) filter. The task consumes input data from array A[i] and produces output data stored in B[i], where $B[i] = C0 * A[i] + C1 * A[i+1] + C2 * A[i+2] + C3 * A[i+3] + C4 * A[i+4]$ and all data elements are 32 bits wide. The task implementation described using a hardware description language (HDL) (FIR.vhd) is synthesized by commercial CAD tools that produce the partial bitstream file (FIR.bit) along with the additional synthesis results for that task. The bitstream contains the configuration data that should be loaded into the configuration memory to instantiate the task at a certain location on the FPGA fabric. The synthesis results are used to determine the rectangle area consumed by the task in terms of configurable logic blocks (CLBs) specified by the width and the height of the task. In our example, the FIR task width is 33 and the task height 32 CLBs for Xilinx Virtex-4 technology. Based on the synthesis output we determine the tasks reconfiguration times. Please note, that in a realistic scenario one additional design space exploration step can be added to steer task shapes toward an optimal point. At such stage, both, task sizes and reconfiguration times are predicted by using high-level models as the ones described in [32] in order to perform quick simulation of the different cases without the need of synthesizing all of the explored task variants. For example in Virtex-4 FPGA technology from Xilinx, there are 22 frames per column and each frame contains 1312 bits. Therefore one column uses $22 \times 1312 = 28{,}864$ bits. Since our FIR hardware task requires 33 CLBs in 2 rows of 16 CLBs totaling in 32 CLBs, we obtain a bitstream with $33 \times 2 \times 28{,}864 = 1{,}905{,}024$ bits. Virtex-4 ICAP can send 32-bit data every 100 MHz clock cycle, hence, we can calculate the reconfiguration time as $1{,}905{,}024 \times 10/32 = 595{,}320$ ns. Next, the FIR task is tested by the designer to determine how fast the input data can be processed. For our example from Fig. 3, the task needs 1415 cycles to process 100, 32-bit input data elements at 11 ns clock period making its ETPUD $1415 \times 11 = 15{,}565$ ns per 100, 32-bit unit data. Based on the above ETPUD number, we can compute the task execution time for various input data sizes. In our example, there are 5000, 32-bit input data elements that have to be processed by the FIR hardware task. Therefore, the expected execution time of our FIR task is $(5000/100) \times 15{,}565 = 778{,}250$ ns (778 ms). The configuration data, together with task specific information, forms a *Task Configuration Microcode* (TCM) block as shown in the upper part of Fig. 3. TCM is pre-stored in memory at the Bitstream (BS) Address. The first field, the BS length represents the size of the configuration data field. This value is used by the Loader when the task is fetched from memory. The task parameter address (TPA) is needed to define where the task input/output parameters are located. In Fig. 3, the task parameters are the input and output data locations, the number of data elements to be processed and the FIR filter coefficients (C0–C4). The input data address gives the location where the data to be processed remains. The location where the output data should be stored is defined by the output data address.

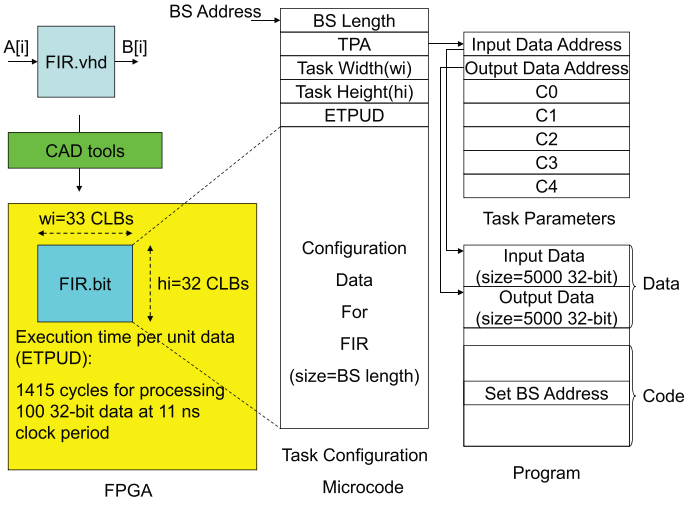

Fig. 3. System at design time.

Please cite this article in press as: Marconi T. Online scheduling and placement of hardware tasks with multiple variants on dynamically reconfigurable field-programmable gate arrays. Comput Electr Eng (2013), http://dx.doi.org/10.1016/j.compeleceng.2013.07.004

During runtime, when the system needs to execute the FIR hardware task on the FPGA, the OS invokes the scheduling and placement algorithm (S&P) to find a location for it (shown as example, in Fig. 4). From the TCM the algorithm gets the task properties information: task width, task height, reconfiguration time, and its execution time. Based on the current state of the used area model, the algorithm searches for the best location for the task. When the location is found (in this example: r' and c'), the task is placed in the area model and its state is updated as shown on the left side of the figure. The Translator converts this location into real physical location on the targeted FPGA. In the bitstream file (FIR.bit), there is information about the FPGA location where the hardware task was pre-designed (in this figure: r and c). By modifying this information at runtime, the Loader partially reconfigures the FPGA through the ICAP (by using the technology specific commands) at the location obtained from the Translator (r'' and c'' in this example). By decoupling our area model from the fine-grain details of the physical FPGA fabric, we propose an FPGA technology independent environment, where different FPGA vendors, e.g., Xilinx, Altera, etc, can provide their consumers with full access to partial reconfigurable resources without exposing all of the proprietary details of the underlying bitstream formats. On the other side, reconfigurable system designers can now focus on partial reconfiguration algorithms without having to bother with all low-level details of a particular FPGA technology.

5. Algorithm

In this section, we discuss the proposed algorithm in detail. We first present the pertinent preliminaries along with some essential supported examples that will serve as foundation to more clearly describe the key mechanisms of the proposed algorithm. Then, to clearly illustrate the basic idea of our proposal, we show a motivational example. Finally, we describe the process of building our proposed strategy based on previous foundational discussions.

5.1. Preliminaries

Definition 1. The acceptable region inside FPGA (W^{FPGA}, H^{FPGA}) of an arriving hardware task $AT(a^{AT}, w^{AT}, h^{AT}, lt^{AT}, d^{AT})$, denoted as AR_{AT}^{FPGA}, is a region where the algorithm can place the *origin* of the arriving task AT without exceeding the FPGA boundary. Therefore, the acceptable region has the lower-leftmost corner $(1,1)$ and upper-rightmost corner $(W^{FPGA} - w^{AT} + 1, H^{FPGA} - h^{AT} + 1)$ as illustrated in Fig. 5.

Definition 2. The conflicting region of an arriving hardware task $AT(a^{AT}, w^{AT}, h^{AT}, lt^{AT}, d^{AT})$ with respect to a scheduled task $ST\left(x_1^{ST}, y_1^{ST}, x_2^{ST}, y_2^{ST}, t_s^{ST}, t_f^{ST}\right)$, denoted as CR_{AT}^{ST}, is the region where the algorithm cannot place the *origin* of the arriving task AT without conflicting with the corresponding scheduled task ST. The conflicting region is defined as the region with its lower-leftmost corner $\left(max(1, x_1^{ST} - w^{AT} + 1), max\left(1, y_1^{ST} - h^{AT} + 1\right)\right)$ and upper-rightmost corner $(min\left(x_2^{ST}, W^{FPGA} - w^{AT} + 1\right), min\left(y_2^{ST}, H^{FPGA} - h^{AT} + 1\right))$ as presented in Fig. 6.

Definition 3. An arriving task $AT(a^{AT}, w^{AT}, h^{AT}, lt^{AT}, d^{AT})$ at position (x_1^{AT}, y_1^{AT}) is said to have a common surface with $FPGA(W^{FPGA}, H^{FPGA})$ boundary if $(x_1^{AT} = 1) \vee \left(x_1^{AT} = W^{FPGA} - w^{AT} + 1\right) \vee (y_1^{AT} = 1) \vee \left(y_1^{AT} = H^{FPGA} - h^{AT} + 1\right)$.

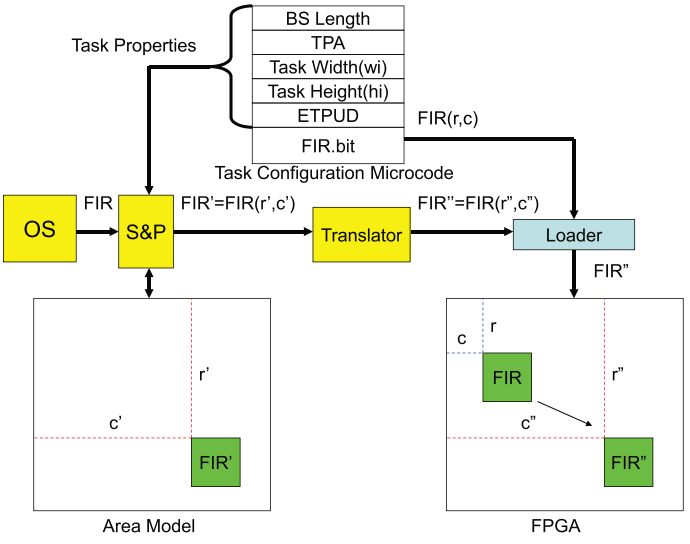

Fig. 4. System at runtime.

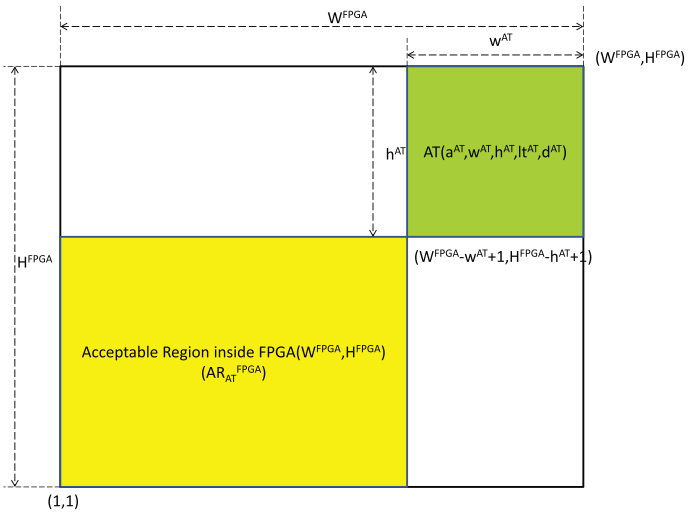

Fig. 5. An example of acceptable region inside $FPGA(W^{FPGA},H^{FPGA})$ of an arriving task $AT(a^{AT},w^{AT},h^{AT},lt^{AT},\ d^{AT})$.

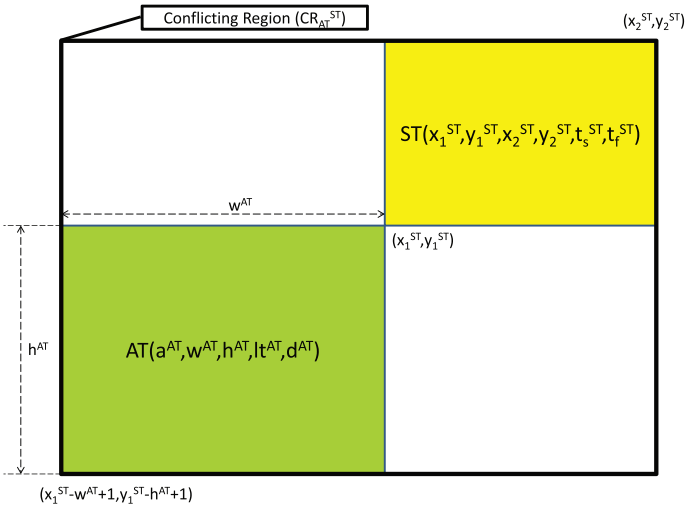

Fig. 6. An example of the conflicting region of an arriving task $AT(a^{AT},w^{AT},h^{AT},lt^{AT},d^{AT})$ with respect to a scheduled task $ST\left(x_1^{ST},y_1^{ST},x_2^{ST},y_2^{ST},t_s^{ST},t_f^{ST}\right)$.

Definition 4. The compaction value with $FPGA(W^{FPGA},H^{FPGA})$ boundary for an arriving task $AT(a^{AT},w^{AT},h^{AT},lt^{AT},d^{AT})$, denoted as CV_{FPGA}^{AT}, is the common surfaces between FPGA boundary and the arriving task.

Example 1. Let us compute CV_{FPGA}^{AT} at position (1,1) of an example in Fig. 7. The arriving task $AT(a^{AT},w^{AT},h^{AT},lt^{AT},d^{AT})$ at position (1,1) has two common surfaces (i.e. bottom and left surfaces) with $FPGA(W^{FPGA},H^{FPGA})$ boundary. Therefore, the CV_{FPGA}^{AT} is the sum area of these surfaces, CV_{FPGA}^{AT} = area of bottom surface + area of left surface = $w^{AT}lt^{AT} + h^{AT}lt^{AT} = (w^{AT} + h^{AT})lt^{AT}$.

Definition 5. An arriving task $AT(a^{AT},w^{AT},h^{AT},lt^{AT},d^{AT})$ at position (x_1^{AT},y_1^{AT}) is said to have a common surface with $ST\left(x_1^{ST},y_1^{ST},x_2^{ST},y_2^{ST},t_s^{ST},t_f^{ST}\right)$ if $\left(\left(t_f^{ST}>t_s^{AT}\right)\wedge(t_s^{ST}\leqslant t_s^{AT})\right)\wedge\neg\left(\left(y_1^{AT}+h^{AT}<y_1^{ST}\right)\vee(y_1^{AT}>y_2^{ST}+1)\vee(x_1^{AT}+w^{AT}<x_1^{ST})\vee\ (x_1^{AT}>x_2^{ST}+1)\right)$.

Definition 6. The compaction value with an scheduled task $ST\left(x_1^{ST},y_1^{ST},x_2^{ST},y_2^{ST},t_s^{ST},t_f^{ST}\right)$ for an arriving task $AT(a^{AT},w^{AT},h^{AT},lt^{AT},d^{AT})$, denoted as CV_{ST}^{AT}, is the common surface between the scheduled task and the arriving task.

Definition 7. The total compaction value of an arriving task $AT(a^{AT},w^{AT},h^{AT},lt^{AT},d^{AT})$, denoted as TCV_{ST}^{AT}, is the sum of CV_{ST}^{AT} with all scheduled tasks that have the common surface with the arriving task AT.

Please cite this article in press as: Marconi T. Online scheduling and placement of hardware tasks with multiple variants on dynamically reconfigurable field-programmable gate arrays. Comput Electr Eng (2013), http://dx.doi.org/10.1016/j.compeleceng.2013.07.004

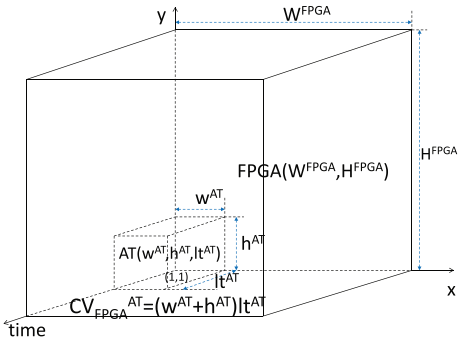

Fig. 7. An example of the compaction value with $FPGA(W^{FPGA},H^{FPGA})$ boundary for an arriving task $AT(a^{AT},w^{AT},h^{AT},lt^{AT},d^{AT})$ at position (1,1).

Example 2. Let us compute CV_{ST}^{AT} at position as illustrated in Fig. 8. The arriving task $AT(a^{AT},w^{AT},h^{AT},lt^{AT},d^{AT})$ at that position has the common surface (i.e. between the right surface of the scheduled task ST and the left surface of the arriving task AT). Therefore, the CV_{ST}^{AT} is the area of this surface, $CV_{ST}^{AT} = h^{AT} cdot \left(t_f^{ST} - t_s^{AT}\right)$.

Definition 8. The finishing time difference with a scheduled task $ST\left(x_1^{ST},y_1^{ST},x_2^{ST},y_2^{ST},t_s^{ST},t_f^{ST}\right)$ for an arriving task $AT(a^{AT},w^{AT},h^{AT},lt^{AT},d^{AT})$, denoted as FTD_{ST}^{AT}, is formulated as $\left|t_f^{AT} - t_f^{ST}\right|$ if the arriving task AT has a common surface with the scheduled task ST.

Definition 9. The total finishing time difference of an arriving task $AT(a^{AT},w^{AT},h^{AT},lt^{AT},d^{AT})$, denoted as $TFTD_{ST}^{AT}$, is the sum of FTD_{ST}^{AT} with all scheduled tasks that have the common surface with the arriving task AT.

Definition 10. The hiding value of an arriving task $AT(a^{AT},w^{AT},h^{AT},lt^{AT},d^{AT})$ at position (x_1^{AT},y_1^{AT}) with a scheduled task $ST\left(x_1^{ST},y_1^{ST},x_2^{ST},y_2^{ST},t_s^{ST},t_f^{ST}\right)$, denoted as HV_{ST}^{AT}, is formulated as $HV_{ST}^{AT} = max(min(x_1^{AT} + w^{AT} - 1, x_2^{ST}) - max(x_1^{AT}, x_1^{ST}) + 1, 0) \times max\left(min\left(y_1^{AT} + h^{AT} - 1, y_2^{ST}\right) - max(y_1^{AT}, y_1^{ST}) + 1, 0\right)$ if $\left(t_f^{ST} = t_s^{AT}\right) \vee \left(t_f^{AT} = t_s^{ST}\right)$.

Example 3. Let us compute the hiding value of an arriving task $AT(a^{AT},w^{AT},h^{AT},lt^{AT},d^{AT})$ at position (x_1^{AT},y_1^{AT}) with a scheduled task $ST\left(x_1^{ST},y_1^{ST},x_2^{ST},y_2^{ST},t_s^{ST},t_f^{ST}\right)$ as illustrated in Fig. 9(a). In this example $t_f^{ST} = t_s^{AT}$, therefore, $HV_{ST}^{AT} = max(min(x_1^{AT} + w^{AT} - 1, x_2^{ST}) - max(x_1^{AT}, x_1^{ST}) + 1, 0) \times max\left(min\left(y_1^{AT} + h^{AT} - 1, y_2^{ST}\right) - max(y_1^{AT}, y_1^{ST}) + 1, 0\right) = max((x_1^{AT} + w^{AT} - 1) - (x_1^{AT}) + 1, 0) \times max((y_2^{ST}) - (y_1^{AT}) + 1, 0) = w^{AT} \times (y_2^{ST} - y_1^{AT} + 1)$.

Example 4. Let us compute the hiding value of an arriving task $AT(a^{AT},w^{AT},h^{AT},lt^{AT},d^{AT})$ at position (x_1^{AT},y_1^{AT}) with a scheduled task $ST\left(x_1^{ST},y_1^{ST},x_2^{ST},y_2^{ST},t_s^{ST},t_f^{ST}\right)$ as illustrated in Fig. 9(b). In this example $t_f^{AT} = t_s^{ST}$, therefore, $HV_{ST}^{AT} = max(min$

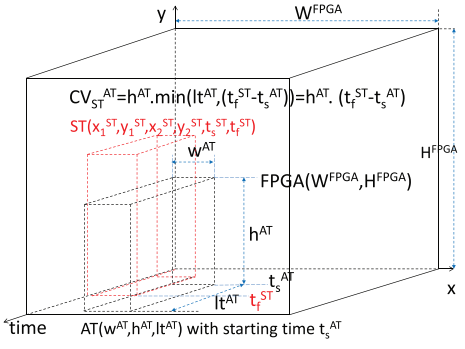

Fig. 8. An example of the compaction value with scheduled task $ST\left(x_1^{ST},y_1^{ST},x_2^{ST},y_2^{ST},t_s^{ST},t_f^{ST}\right)$ for an arriving task $AT(a^{AT},w^{AT},h^{AT},lt^{AT},d^{AT})$.

Please cite this article in press as: Marconi T. Online scheduling and placement of hardware tasks with multiple variants on dynamically reconfigurable field-programmable gate arrays. Comput Electr Eng (2013), http://dx.doi.org/10.1016/j.compeleceng.2013.07.004

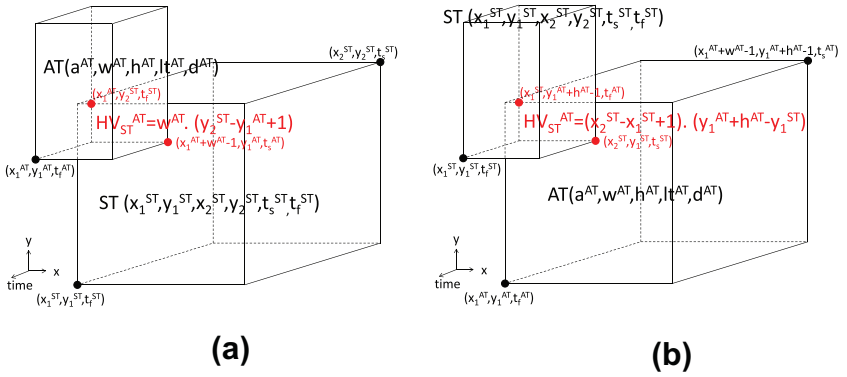

Fig. 9. Examples of the hiding value of an arriving task $AT(a^{AT},w^{AT},h^{AT},lt^{AT},d^{AT})$ at position (x_1^{AT},y_1^{AT}) with a scheduled task $ST\left(x_1^{ST},y_1^{ST},x_2^{ST},y_2^{ST},t_s^{ST},t_f^{ST}\right)$.

$$(x_1^{AT}+w^{AT}-1,x_2^{ST})-max(x_1^{AT},x_1^{ST})+1,0)\times max\left(min\left(y_1^{AT}+h^{AT}-1,y_2^{ST}\right)-max(y_1^{AT},y_1^{ST})+1,0\right) = max((x_2^{ST})-(x_1^{ST})+1,0)\times max\left(\left(y_1^{AT}+h^{AT}-1\right)-(y_1^{ST})+1,0\right)=(x_2^{ST}-x_1^{ST}+1)\times\left(y_1^{AT}+h^{AT}-y_1^{ST}\right).$$

Definition 11. The total hiding value of an arriving task $AT(a^{AT},w^{AT},h^{AT},lt^{AT},d^{AT})$, denoted as THV_{ST}^{AT}, is the sum of HV_{ST}^{AT} with all scheduled tasks that satisfy the $\left(t_f^{ST}=t_s^{AT}\right)\vee\left(t_f^{AT}=t_s^{ST}\right)$ constraint.

Definition 12. The starting time matrix of an arriving task $AT(a^{AT},w^{AT},h^{AT},lt^{AT},d^{AT})$ on $FPGA(W^{FPGA},H^{FPGA})$, denoted as STM(x,y), is a $(W^{FPGA}-w^{AT}+1)\times(H^{FPGA}-h^{AT}+1)$ matrix in which the element of STM(i,j) of the matrix is the earliest starting time for the arriving task AT placed on the FPGA at the potential position column i and row j.

Example 5. Let us compute the starting time matrix of an arriving task $AT(2,3,3,5,10)$ on $FPGA(10,10)$ in Fig. 10. The size of STM(x,y) is $(W^{FPGA}-w^{AT}+1)\times(H^{FPGA}-h^{AT}+1)=(10-3+1)\times(10-3+1)=8\times8$. Let us assume that the current time is 2 time units.

To computer the value of STM$(1,1)$, we have to place the arriving task on the FPGA at the potential position column 1 and row 1. Since the size of the arriving task is 3×3, if we place this arriving task at position $(1,1)$, we need to use reconfigurable units from columns 1 to 3 and rows 1 to 3. In this current time, these needed reconfigurable units for the arriving task are being used by the scheduled task ST_1. Therefore, the earliest starting time at position $(1,1)$ for the AT will be after the ST_1 terminates at time unit $t_f^{ST_1}=9$, STM$(1,1)=9$.

To computer the value of STM$(1,4)$, we have to place the arriving task on the FPGA at the potential position column 1 and row 4. If we place this arriving task at this position, we need to use reconfigurable units from columns 1 to 3 and rows 4 to 6. In this current time, these needed reconfigurable units for the arriving task are being used partially by the scheduled task ST_1

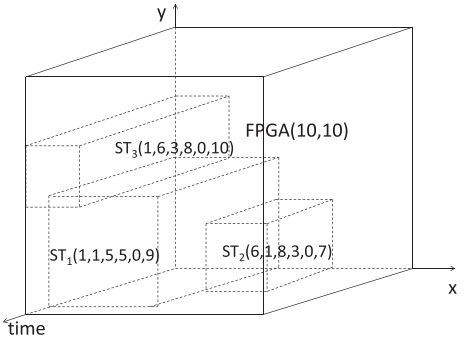

Fig. 10. An example for the computation of the starting time matrix.

(columns 1–3 and rows 4–5) and ST_3 (columns 1–3 and row 6). Therefore, the earliest starting time at this position for the AT will be after both ST_1 and ST_3 terminate at time unit $max\left(t_f^{ST_1}, t_f^{ST_3}\right) = max(9, 10) = 10$, STM(1,4) = 10.

Using the same method as above computations, we can compute the values of all elements in the STM(x,y) matrix to produce the STM(x,y) matrix as following:

$$\mathrm{STM}(x,y) = \begin{bmatrix} 9 & 9 & 9 & 10 & 10 & 10 & 10 & 10 \\ 9 & 9 & 9 & 10 & 10 & 10 & 10 & 10 \\ 9 & 9 & 9 & 10 & 10 & 10 & 10 & 10 \\ 9 & 9 & 9 & 9 & 9 & 2 & 2 & 2 \\ 9 & 9 & 9 & 9 & 9 & 2 & 2 & 2 \\ 7 & 7 & 7 & 2 & 2 & 2 & 2 & 2 \\ 7 & 7 & 7 & 2 & 2 & 2 & 2 & 2 \\ 7 & 7 & 7 & 2 & 2 & 2 & 2 & 2 \end{bmatrix}$$

5.2. Motivational example

The basic idea of our proposal is to use as least as possible reconfigurable resources to meet the deadline constraint of each incoming task. In order to clearly describe the main idea of our proposal, we use a motivational example (Fig. 11) which sketches the scheduling and placement of a hardware task set by single mode task algorithms and multi mode task algorithms. In this figure, the length and the width of a rectangle represents the required reconfigurable area of FPGA to run the corresponding task and its lifetime of staying on this fabric.

Fig. 11(a) shows a simple hardware task set that is composed of 3 single mode hardware tasks (T_1, T_2, and T_3); each task has only one hardware version (i.e. $T_{1,2}$, $T_{2,2}$, or $T_{3,2}$). Unlike the single mode hardware task set, each hardware task in the multi mode hardware task set is composed of a variety of different hardware implementations as illustrated in Fig. 11(b). In this instance, each task has two hardware versions depending on the desired performance (e.g. task T_1 with its two hardware versions $T_{1,1}$ and $T_{1,2}$). Each version has its own speed with required reconfigurable area for running it on FPGA. For example, with more required area, version $T_{1,2}$ runs faster than version $T_{1,1}$. Let us assume that these three tasks arrive to the system at time t = 0 and the FPGA has 17 free reconfigurable units at that time. Single mode task algorithms allocate task $T_{1,2}$ at the left side of the FPGA (units 1–10), starting from t = 0 and ending at t = 8; task T_1 meets its deadline as shown in Fig. 11(c)-i. Instead of using $T_{1,2}$, multi mode task algorithms decide to run task T_1 using a different hardware version $T_{1,1}$ with slower speed but less required area, which still can meet its deadline constraint as illustrated in Fig. 11(c)-ii. The idea is that this saved area can be used later by other tasks. For task T_2, the multi mode algorithms do not use a smaller hardware version ($T_{2,1}$), because this hardware version cannot meet the deadline constraint of task T_2. Therefore, the larger hardware version $T_{2,2}$ is chosen for running task T_2 that can satisfy the deadline constraint as shown in Fig. 11(c)-ii. Since single mode task algorithms is paralyzed by its inflexibility of choosing the best solution to run tasks in reconfigurable fabric, almost all reconfigurable area at time t = 0 is occupied by other tasks (only left 1 unit), task T_3 cannot be operated at time t = 0. Therefore, it needs to be scheduled later, in this case at t = 14, after the termination of $T_{2,2}$. As a consequence, it cannot meet its deadline as shown in Fig. 11(c)-i. This miss deadline case can be eliminated if we use multi mode algorithms. In the multi mode algorithm, since it saves a lot area during scheduling previous task (in this case task T_1), free area at time t = 0 still big enough for allocating hardware version $T_{3,1}$. As a result, the deadline constraint of T_3 is met as illustrated in Fig. 11(c)-ii.

5.3. Pseudocode and analysis

By orchestrating all heuristics introduced earlier, we develop the proposed strategy, called as *pruning moldable (PM)*; the pseudocode of the algorithm is shown in Algorithm 1. To reserve more free area for future incoming tasks, when a new task with multiple hardware versions AT arrives to the system, the algorithm tries to choose the slowest hardware task first in which the required reconfigurable area is minimized as shown in line 1. With respect to this chosen hardware implementation, the starting time matrix STM(x,y) adopted from Definition 12 is built as shown in line 2. This step will be explained further later when we are talking about Algorithm 2. The key idea of this step is to empower the algorithm to always choose the earliest starting time for every single arriving task. In line 3, the algorithm finds the *smallest starting time* (SST) from STM(x,y), which is the smallest value in the matrix. This SST is the earliest time we can schedule the chosen hardware task, which is optimized in time direction. This best position in time will be utilized later in line 4 to steer the algorithm in making decision of the best hardware variant. This result also will be used further to steer the algorithm to choose the best position not only in time but also in space directions later in lines 8–15 as our goals.

The algorithm is empowered with an automatic adaptation to choose the best hardware version on-the-fly among many alternative versions to adapt its state-of-the-art FPGA resources to its current real time requirement at runtime. The key to this is to run each incoming request in fulfilling its deadline constraint with as small as possible of reconfigurable hardware resources. Since it should adapt at runtime with the current condition of FPGA resources, the same hardware task can be treated differently. Depending on the type of request sent from the user and the dynamic condition of FPGA resources recognized by the algorithm, the algorithm responds in different ways. To implement this idea, in line 4, the algorithm checks

Please cite this article in press as: Marconi T. Online scheduling and placement of hardware tasks with multiple variants on dynamically reconfigurable field-programmable gate arrays. Comput Electr Eng (2013), http://dx.doi.org/10.1016/j.compeleceng.2013.07.004

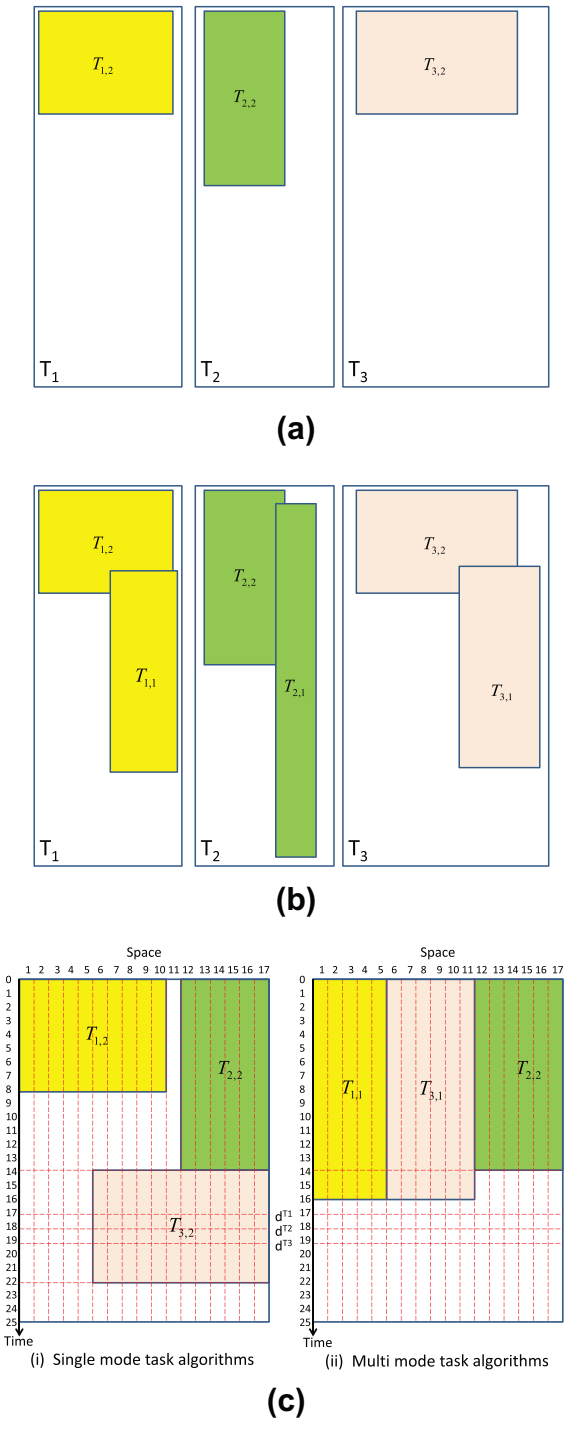

Fig. 11. Scheduling and placement of task set in (a) and (b) by single mode algorithms (c)-i versus multi mode algorithms (c)-ii.

the current solution. If the current solution cannot satisfy the deadline constraint of the respective task (i.e. $t_f^{AT} > d^{AT}$) and all implemented hardwares have not been tried yet, the algorithm switches to the faster hardware variant with more hardware resources for finding a better solution (lines 5–6) looping back to line 2 for computing a new STM(x,y) matrix with respect to the new hardware version. Otherwise, it skips lines 5–6 and proceeds to line 8 to start finding the best solution among best candidates.

To significantly reduce runtime overhead, the search space should be minimized by avoiding any unnecessary search. For that purpose, the algorithm is armed with an ability to prune search space. With this ability, it is not necessary to check all

Please cite this article in press as: Marconi T. Online scheduling and placement of hardware tasks with multiple variants on dynamically reconfigurable field-programmable gate arrays. Comput Electr Eng (2013), http://dx.doi.org/10.1016/j.compeleceng.2013.07.004

elements in STM(x,y) matrix for searching the best position for each incoming task. It only needs to check the potential positions (line 8) that can satisfy the *pruning constraints* (line 9) using its *2D-directional pruning capability* (Fig. 12) from the best positions steered by SST from line 3. Line 8 gives the algorithm an ability to choose the only best potential positions from all positions in STM(x,y) matrix in time direction. The search space is pruned further by only searching *unrepeated adjacent elements* in x direction $((STM(x,y) = SST \wedge STM(x-1,y) \neq SST \wedge x \neq 1) \vee (STM(x,y) = SST \wedge STM(x+1,y) \neq SST \wedge x \neq W^{FPGA} - w^{AT} + 1))$ and y direction $((STM(x,y) = SST \wedge STM(x,y-1) \neq SST \wedge y \neq 1) \vee (STM(x,y) = SST \wedge STM(x,y+1) \neq SST \wedge y \neq H^{FPGA} - h^{AT} + 1))$ and *FPGA boundary adjacent elements* $((STM(x,y) = SST \wedge x = 1) \vee (STM(x,y) = SST \wedge x = W^{FPGA} - w^{AT} + 1) \vee (STM(x,y) = SST \wedge y = 1) \vee (STM(x,y) = SST \wedge y = H^{FPGA} - h^{AT} + 1))$ within the area dictated by line 8 as shown in Fig. 12. By doing this, the algorithm can find the best solution for each arriving task faster as will be shown later in the evaluation section.

After pruning the searching space, the algorithm needs to find the best position for the incoming task among best potential positions produced by previous pruning processes. To do this, the variable α^{AT} is introduced in line 12, which is the sum of CV^{AT}_{FPGA} (Definition 4), TCV^{AT}_{ST} (Definition 7), and THV^{AT}_{ST} (Definition 11). This new variable is used in line 13 to advise the algorithm to find the best solution. The computation of CV^{AT}_{FPGA}, TCV^{AT}_{ST}, THV^{AT}_{ST}, and $TFTD^{AT}_{ST}$ is done in lines 10–11. In line 13, the algorithm chooses the best position for the chosen hardware version based on its current availability of hardware resources, resulting the best position both in space and time.

Algorithm 1. Pruning Moldable (PM)

1: choose the slowest hardware version
2: build STM(x,y) (see Algorithm 2)
3: find the smallest starting time (SST)
4: **if** $t_f^{AT} > d^{AT}$ **and** all alternative hardware versions have not been tried
5: switch to the faster hardware version
6: go to line 2
7: **end if**
8: **for** all positions with SST **do**
9: **if** (STM(x,y) = SST **and** $x = 1$) **or**
(STM(x,y) = SST **and** STM($x-1,y$) ≠ SST **and** $x \neq 1$) **or**
(STM(x,y) = SST **and** STM($x+1,y$) ≠ SST **and** $x \neq W^{FPGA} - w^{AT} + 1$) **or**
(STM(x,y) = SST **and** $x = W^{FPGA} - w^{AT} + 1$) **or**
(STM(x,y) = SST **and** $y = 1$) **or**
(STM(x,y) = SST **and** STM($x,y-1$) ≠ SST **and** $y \neq 1$) **or**
(STM(x,y) = SST **and** STM($x,y+1$) ≠ SST **and** $y \neq H^{FPGA} - h^{AT} + 1$) **or**
(STM(x,y) = SST **and** $y = H^{FPGA} - h^{AT} + 1$)
10: compute CV^{AT}_{FPGA} (see Algorithm 3)
11: compute TCV^{AT}_{ST}, THV^{AT}_{ST}, and $TFTD^{AT}_{ST}$ (see Algorithm 4)
12: $\alpha^{AT} = CV^{AT}_{FPGA} + TCV^{AT}_{ST} + THV^{AT}_{ST}$
13: choose the best position (see Algorithm 5)
14: **end if**
15: **end for**
16: update linked lists (see Algorithm 6)

The detail information of how the algorithm builds the STM(x,y) matrix is shown in Algorithm 2. The aim is to quickly find the earliest time for launching each newly-arrived task by inspecting its acceptable and conflicting regions (Definitions 1 and 2). The constructing of STM(x,y) matrix consists of an initialization and a series of updating operations. Two linked lists, denoted as the execution list EL and the reservation list RL, are managed by the algorithm. The EL keeps track of all currently running tasks on FPGA, sorted in order of increasing finishing times; the RL records all scheduled tasks, sorted in order of increasing starting times. Initially, there are no tasks in the lists (EL = ∅ and RL = ∅). Each element in the lists holds the information of the stored task: the *lower-left corner* (x_1,y_1), the *upper-right corner* (x_2,y_2), the *starting time* t_s, the *finishing time* t_f, the task name, the next pointer, and the previous pointer. Recalling to Definition 12, each element of STM(x,y) matrix contains the earliest starting time for the arriving task at the potential position directed by variables x and y. The basic idea of constructing STM(x,y) matrix is to initialize the matrix by the arrival time of the incoming task (lines 1–5) and to update its contents with respect to scheduled tasks both in the execution and reservation lists (lines 6–23). In lines 1–2, the size of STM(x,y) matrix is equal to the size of the acceptable region AR^{FPGA}_{AT} as defined in Definition 1. If there is no task in linked lists (EL=∅ and RL=∅), all elements of the matrix will be equal to the arrival time of the incoming task a^{AT} (line 3). In case of there is at least one task in the EL (EL ≠ ∅) as shown in line 6, the algorithm needs to check whether this set of tasks overlaps in space with the arriving task. This space overlapping check in lines 7–8 can be done by utilizing the definition of conflicting region CR^{ST}_{AT} as discussed in

Please cite this article in press as: Marconi T. Online scheduling and placement of hardware tasks with multiple variants on dynamically reconfigurable field-programmable gate arrays. Comput Electr Eng (2013), http://dx.doi.org/10.1016/j.compeleceng.2013.07.004

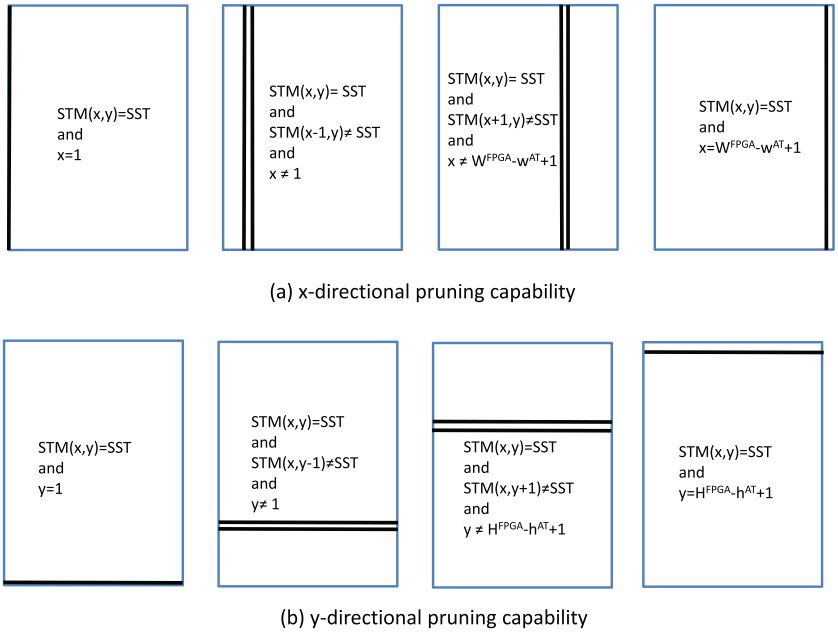

Fig. 12. 2D-directional pruning capability.

Definition 2. Hence, by inspecting the acceptable and conflicting regions, it is not necessary to check the state of each reconfigurable unit in the FPGA. Not all elements in this overlapped area need to be updated to the finishing time of scheduled task t_f^{ST}; only elements that have values less than t_f^{ST} need to be updated (line 9). The STM(x,y) matrix also can be influenced by tasks in the reservation list RL if RL $\neq \emptyset$ (line 15). Similar like updating STM(x,y) by the EL, the area defined by CR_{AT}^{ST} needs to be evaluated (lines 16–17). Besides checking the finishing time, the updating of STM(x,y) by the RL needs to check the starting time of scheduled tasks as well (line 18). This process needs to ensure that only the element overlapped in space and time with scheduled tasks needs to be updated (line 19).

Let us use this algorithm to compute STM(x,y) matrix in Example 5 and Fig. 10. From Fig. 10, since the current time is two time units, all tasks in the figure are under operation; as a result, all tasks are recorded in the execution list sorted in order of increasing finishing times (i.e. $EL = \{ST_3, ST_1, ST_2\}$) and there is no task in the reservation list. Since the $RL = \emptyset$, the execution of lines 15–23 does not affect the STM(x,y) matrix. After execution of lines 1–5, the STM(x,y) will be filled with the arrival time of the incoming task AT(2,3,3,5,5,10) (i.e. $a^{AT} = 2$ time units); hence, the matrix will be as following:

$$\mathrm{STM}(x,y) = \begin{bmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{bmatrix}$$

Next, the algorithm proceeds to lines 6–14, updating STM(x,y) with respect to scheduled tasks in the execution list $EL = \{ST_2, ST_1, ST_3\}$, sorted in order of increasing finishing times. Since the first scheduled task is ST_2, after running lines 6–14 for ST_2, the STM(x,y) affected by ST_2 will be updated by its finishing time (i.e. $t_f^{ST_2} = 7$ time units). As a result, the STM(x,y) will be as following:

$$\mathrm{STM}(x,y) = \begin{bmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 7 & 7 & 7 & 2 & 2 & 2 & 2 & 2 \\ 7 & 7 & 7 & 2 & 2 & 2 & 2 & 2 \\ 7 & 7 & 7 & 2 & 2 & 2 & 2 & 2 \end{bmatrix}$$

Please cite this article in press as: Marconi T. Online scheduling and placement of hardware tasks with multiple variants on dynamically reconfigurable field-programmable gate arrays. Comput Electr Eng (2013), http://dx.doi.org/10.1016/j.compeleceng.2013.07.004

Subsequently, using the same method, the algorithm needs to update the STM(x,y) affected by ST_1 (i.e. $t_f^{ST_1} = 9$ time units) as following:

$$\mathrm{STM}(x,y) = \begin{bmatrix} 9 & 9 & 9 & 9 & 9 & 2 & 2 & 2 \\ 9 & 9 & 9 & 9 & 9 & 2 & 2 & 2 \\ 9 & 9 & 9 & 9 & 9 & 2 & 2 & 2 \\ 9 & 9 & 9 & 9 & 9 & 2 & 2 & 2 \\ 9 & 9 & 9 & 9 & 9 & 2 & 2 & 2 \\ 7 & 7 & 7 & 2 & 2 & 2 & 2 & 2 \\ 7 & 7 & 7 & 2 & 2 & 2 & 2 & 2 \\ 7 & 7 & 7 & 2 & 2 & 2 & 2 & 2 \end{bmatrix}$$

Finally, again using the same strategy, the algorithm proceeds to update the STM(x,y) affected by ST_3 (i.e. $t_f^{ST_3} = 10$ time units) as following:

$$\mathrm{STM}(x,y) = \begin{bmatrix} 9 & 9 & 9 & 10 & 10 & 10 & 10 & 10 \\ 9 & 9 & 9 & 10 & 10 & 10 & 10 & 10 \\ 9 & 9 & 9 & 10 & 10 & 10 & 10 & 10 \\ 9 & 9 & 9 & 9 & 9 & 2 & 2 & 2 \\ 9 & 9 & 9 & 9 & 9 & 2 & 2 & 2 \\ 7 & 7 & 7 & 2 & 2 & 2 & 2 & 2 \\ 7 & 7 & 7 & 2 & 2 & 2 & 2 & 2 \\ 7 & 7 & 7 & 2 & 2 & 2 & 2 & 2 \end{bmatrix}$$

Algorithm 2. Building STM(x,y) matrix

```
1: for (x = 1; x ⩽ W^FPGA − w^AT + 1; x++) do
2:   for (y = 1; y ⩽ H^FPGA − h^AT + 1; y++) do
3:     STM(x,y) = a^AT
4:   end for
5: end for
6: for all tasks in the EL do
7:   for (x = max(1,x_1^ST − w^AT + 1); x ⩽ min(x_2^ST, W^FPGA − w^AT + 1); x++) do
8:     for (y = max(1,y_1^ST − h^AT + 1); y ⩽ min(y_2^ST, H^FPGA − h^AT + 1); y++) do
9:       if STM(x,y) < t_f^ST then
10:          STM(x,y) = t_f^ST
11:      end if
12:    end for
13:  end for
14: end for
15: for all tasks in the RL do
16:  for (x = max(1,x_1 − w^AT + 1); x ⩽ min(x_2, W^FPGA − w^AT + 1); x++) do
17:    for (y = max(1,y_1 − h^AT + 1); y ⩽ min(y_2, H^FPGA − h^AT + 1); y++) do
18:      if (STM(x,y) < t_f^ST and STM(x,y) + lt^AT > t_s^ST) then
19:        STM(x,y) = t_f^ST
20:      end if
21:    end for
22:  end for
23: end for
```

Please cite this article in press as: Marconi T. Online scheduling and placement of hardware tasks with multiple variants on dynamically reconfigurable field-programmable gate arrays. Comput Electr Eng (2013), http://dx.doi.org/10.1016/j.compeleceng.2013.07.004

The pseudocode to compute compaction value with FPGA boundary, denoted as CV^{AT}_{FPGA} and adopted from Definition 4, is shown in Algorithm 3. The motivation of this heuristic is to give awareness of FPGA boundary to the algorithm. The CV^{AT}_{FPGA} is zero (line 1) if the position of the arriving task does not satisfy the condition defined in Definition 3 (lines 2, 4, and 6); in this case, the arriving tasks does not have any common surface with FPGA boundary. The initialization of CV^{AT}_{FPGA} is presented in line 1; it is set to zero. The condition in line 2 is satisfied if two surfaces of the arriving task in 3D representation touch the FPGA boundary, i.e. the arriving task is allocated at one of the corners of FPGA; in this situation, the CV^{AT}_{FPGA} is equal to the sum of area of the common surfaces, $(w^{AT} + h^{AT}) \times lt^{AT}$ as shown in line 3. Please note that the 3D representation of hardware task has been discussed in Section 3. The satisfaction of condition either in line 4 or 6 means that the arriving task has only one common surface with FPGA boundary; here, the CV^{AT}_{FPGA} can be calculated as the area of that common surface which is either $h^{AT} \times lt^{AT}$ (line 5) or $w^{AT} \times lt^{AT}$ (line 7).

Let us use this algorithm to compute CV^{AT}_{FPGA} in Example 1 and Fig. 7. First, in line 1, the algorithm initializes CV^{AT}_{FPGA} to zero. Since the potential position is (1,1) (i.e. $x^{AT}_1 = 1$ and $y^{AT}_1 = 1$), the condition in line 2 is satisfied. As a result, the algorithm proceeds from line 2 to line 3 and ends up to line 8. Therefore, the CV^{AT}_{FPGA} is computed as $CV^{AT}_{FPGA} = (w^{AT} + h^{AT}) \times lt^{AT}$.

Algorithm 3. Computing CV^{AT}_{FPGA}

1: $CV^{AT}_{FPGA} = 0$
2: **if** $(x^{AT}_1 = 1$ **and** $y^{AT}_1 = 1)$ **or**
$(x^{AT}_1 = 1$ **and** $y^{AT}_1 = H^{FPGA} - h^{AT} + 1)$ **or**
$(x^{AT}_1 = W^{FPGA} - w^{AT} + 1$ **and** $y^{AT}_1 = H^{FPGA} - h^{AT} + 1)$ **or**
$(x^{AT}_1 = W^{FPGA} - w^{AT} + 1$ **and** $y^{AT}_1 = 1)$ **then**
3: $CV^{AT}_{FPGA} = (w^{AT} + h^{AT}) \times lt^{AT}$
4: **else if** $(x^{AT}_1 = 1$ **and** $y^{AT}_1 \neq 1$ **and** $y^{AT}_1 \neq H^{FPGA} - h^{AT} + 1)$ **or**
$(x^{AT}_1 = W^{FPGA} - w^{AT} + 1$ **and** $y^{AT}_1 \neq 1$ **and** $y^{AT}_1 \neq H^{FPGA} - h^{AT} + 1)$ **then**
5: $CV^{AT}_{FPGA} = h^{AT} \times lt^{AT}$
6: **else if** $(x^{AT}_1 \neq 1$ **and** $y^{AT}_1 = 1$ **and** $x^{AT}_1 \neq W^{FPGA} - w^{AT} + 1)$ **or**
$(x^{AT}_1 \neq 1$ **and** $y^{AT}_1 = H^{FPGA} - h^{AT} + 1$ **and** $x^{AT}_1 \neq W^{FPGA} - w^{AT} + 1)$ **then**
7: $CV^{AT}_{FPGA} = w^{AT} \times lt^{AT}$ 8: **end if**

Algorithm 4 shows the pseodocode of computing TCV^{AT}_{ST} (Definition 7), THV^{AT}_{ST} (Definition 11), and $TFTD^{AT}_{ST}$ (Definition 8). The heuristic TCV^{AT}_{ST} is required to equip the algorithm to pack tasks in time and space. To give the algorithm an awareness of "blocking effect", the heuristic THV^{AT}_{ST} is added to it. The last employed heuristic $TFTD^{AT}_{ST}$ steers the algorithm to group hardware tasks with similar finishing times; as a result, more adjacent tasks depart from FPGA at the same time, creating the larger free area during deallocation. The initialization is done in lines 1–3; all variables are set to zero. To compute the values of these variables, the algorithm needs to check all scheduled tasks in linked lists RL and EL (line 4) that can affect their values. Not all scheduled tasks in linked lists need to be considered for computation, only tasks touched by the arriving task. For example in Fig. 13, the considered tasks are only three tasks (T_{12}, T_{15} and T_{16}). Definition 5 discussed before is implemented

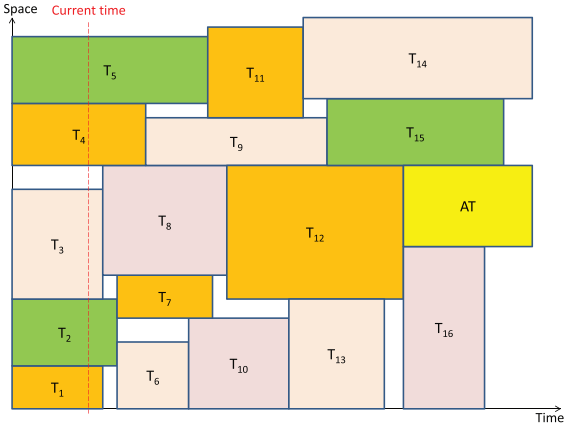

Fig. 13. The only considered tasks (T_{12}, T_{15} and T_{16}) for an arriving task AT at this position.

Please cite this article in press as: Marconi T. Online scheduling and placement of hardware tasks with multiple variants on dynamically reconfigurable field-programmable gate arrays. Comput Electr Eng (2013), http://dx.doi.org/10.1016/j.compeleceng.2013.07.004

Table 1
Time complexity analysis of Pruning Moldable (PM) algorithm.

Line(s)	Time complexity
1	$O(1)$
2	$O(W \times H \times max(N_{ET}, N_{RT}))$
3	$O(W \times H)$
4–7	$O(n \times W \times H \times max(N_{ET}, N_{RT}))$
8–15	$O(W \times H \times max(N_{ET}, N_{RT}))$
16	$O(max(N_{ET}, N_{RT}))$
Total	$O(n \times W \times H \times max(N_{ET}, N_{RT}))$

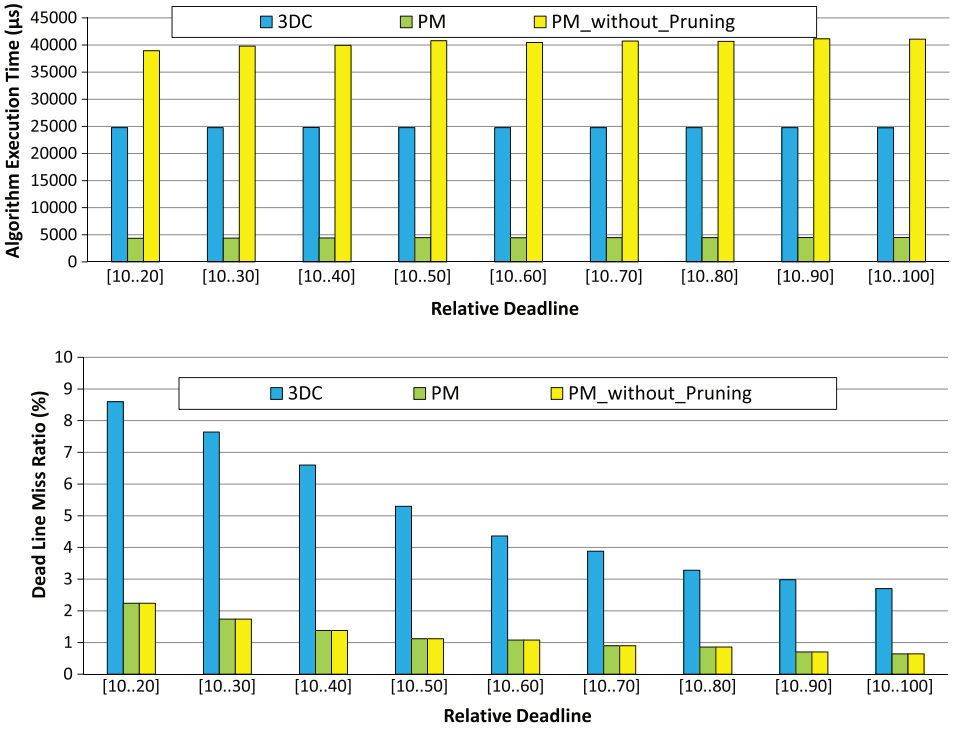

Fig. 14. Evaluation with synthetic hardware tasks.

in line 5 in this algorithm. If Definition 5 is not satisfied, the TCV_{ST}^{AT} and $TFTD_{ST}^{AT}$ are zero. The scheduled task and the arriving task can have a common surface in one of different ways as shown in lines 6–30. Based on Definitions 6 and 7, the TCV_{ST}^{AT} is computed as the sum of common surface by its considered tasks; each added value has two components (commonality in space and time). For example, in line 7, the commonality in space is computed as w^{AT} while the commonality in time is represented as $min\left(lt^{AT}, \left(t_f^{ST} - t_s^{AT}\right)\right)$. As far as there is a common surface between the scheduled task and the arriving task, the FTD_{ST}^{AT} is added cumulatively to $TFTD_{ST}^{AT}$, which is the finishing time difference between the arriving task and considered tasks as defined in Definitions 8 and 9. The computation of total hiding value THV_{ST}^{AT} of Definitions 10 and 11 is applied in lines 31–33; it only has value if one of these conditions is satisfied: (1) the arriving task is scheduled immediately once the scheduled task is terminated ($t_f^{ST} = t_s^{AT}$, e.g. the arriving task AT and task T_{12} in Fig. 13) or (2) the termination of arriving task is right away before the start of the scheduled task $\left(t_f^{AT} = t_s^{ST}\right)$ as shown in line 31.

Let us compute TCV_{ST}^{AT}, THV_{ST}^{AT}, and $TFTD_{ST}^{AT}$ in Fig. 8 using this algorithm. In this situation, the execution list is filled by one scheduled task $ST\left(x_1^{ST}, y_1^{ST}, x_2^{ST}, y_2^{ST}, t_s^{ST}, t_f^{ST}\right)$ (i.e. $EL = \left\{ST(x_1^{ST}, y_1^{ST}, x_2^{ST}, y_2^{ST}, t_s^{ST}, t_f^{ST})\right\}$); whereas the reservation list is empty. All computation variables is initialized to zero in lines 1–3 (i.e. $TCV_{ST}^{AT} = 0$, $THV_{ST}^{AT} = 0$, $TFTD^{AT} = 0$). In line 4, the algorithm considers the scheduled task in EL, $ST(x_1^{ST}, y_1^{ST}, x_2^{ST}, y_2^{ST}, t_s^{ST}, t_f^{ST})$. Since the condition in line 5 and line 9 are satisfied, the algorithm proceeds to lines 10-11 and ends up at line 34. In line 10, the total compaction value with scheduled tasks is updated,

Please cite this article in press as: Marconi T. Online scheduling and placement of hardware tasks with multiple variants on dynamically reconfigurable field-programmable gate arrays. Comput Electr Eng (2013), http://dx.doi.org/10.1016/j.compeleceng.2013.07.004

$TCV_{ST}^{AT} = TCV_{ST}^{AT} + h^{AT} \times min\left(lt^{AT}, \left(t_f^{ST} - t_s^{AT}\right)\right) = 0 + h^{AT} \times min\left(lt^{AT}, \left(t_f^{ST} - t_s^{AT}\right)\right) = h^{AT} \times min\left(lt^{AT}, \left(t_f^{ST} - t_s^{AT}\right)\right)$. After execution line 11, the total finishing time difference is renewed, $TFTD_{ST}^{AT} = TFTD_{ST}^{AT} + |t_f^{AT} - t_f^{ST}| = 0 + |t_f^{AT} - t_f^{ST}| = |t_f^{AT} - t_f^{ST}|$. Since the algorithm skips lines 31-33, the total hiding value becomes zero.

Algorithm 4. Computing TCV_{ST}^{AT}, THV_{ST}^{AT}, and $TFTD_{ST}^{AT}$

1: $TCV_{ST}^{AT} = 0$
2: $THV_{ST}^{AT} = 0$
3: $TFTD^{AT} = 0$
4: **for** all tasks in the RL and EL **do**
5: **if** $(t_f^{ST} > t_s^{AT}$ **and** $t_s^{ST} \leqslant t_s^{AT})$ **and not** $(y_1^{AT} + h^{AT} < y_1^{ST}$ **or** $y_1^{AT} > y_2^{ST} + 1$ **or** $x_1^{AT} + w^{AT} < x_1^{ST}$ **or** $x_1^{AT} > x_2^{ST} + 1)$ **then**
6: **if** $(x_1^{ST} \leqslant x_1^{AT} \leqslant x_2^{ST}$ **and** $x_1^{ST} - w^{AT} + 1 \leqslant x_1^{AT} \leqslant x_2^{ST} - w^{AT} + 1$ **and** $y_1^{AT} = y_1^{ST} - h^{AT})$ **or** $(x_1^{ST} \leqslant x_1^{AT} \leqslant x_2^{ST}$ **and** $x_1^{ST} - w^{AT} + 1 \leqslant x_1^{AT} \leqslant x_2^{ST} - w^{AT} + 1$ **and** $y_1^{AT} = y_2^{ST} + 1)$ **then**
7: $TCV_{ST}^{AT} = TCV_{ST}^{AT} + w^{AT} \times min(lt^{AT}, (t_f^{ST} - t_s^{AT}))$
8: $TFTD_{ST}^{AT} = TFTD_{ST}^{AT} + |t_f^{AT} - t_f^{ST}|$
9: **else if** $(y_1^{ST} \leqslant y_1^{AT} \leqslant y_2^{ST}$ **and** $y_1^{ST} - h^{AT} + 1 \leqslant y_1^{AT} \leqslant y_2^{ST} - h^{AT} + 1$ **and** $x_1^{AT} = x_1^{ST} - w^{AT})$ **or** $(y_1^{ST} \leqslant y_1^{AT} \leqslant y_2^{ST}$ **and** $y_1^{ST} - h^{AT} + 1 \leqslant y_1^{AT} \leqslant y_2^{ST} - h^{AT} + 1$ **and** $x_1^{AT} = x_2^{ST} + 1)$
10: $TCV_{ST}^{AT} = TCV_{ST}^{AT} + h^{AT} \times min(lt^{AT}, (t_f^{ST} - t_s^{AT}))$
11: $TFTD_{ST}^{AT} = TFTD_{ST}^{AT} + |t_f^{AT} - t_f^{ST}|$
12:**else if** $(x_1^{ST} - w^{AT} + 1 \leqslant x_1^{AT} \leqslant x_1^{ST}$ **and** $x_2^{ST} - w^{AT} + 1 \leqslant x_1^{AT} \leqslant x_2^{ST}$ **and** $y_1^{AT} = y_2^{ST} + 1)$ **or** $(x_1^{ST} - w^{AT} + 1 \leqslant x_1^{AT} \leqslant x_1^{ST}$ **and** $x_2^{ST} - w^{AT} + 1 \leqslant x_1^{AT} \leqslant x_2^{ST}$ **and** $y_1^{AT} = y_1^{ST} - h^{AT})$ **then**
13: $TCV_{ST}^{AT} = TCV_{ST}^{AT} + (x_2^{ST} - x_1^{ST} + 1) \times min(lt^{AT}, (t_f^{ST} - t_s^{AT}))$
14: $TFTD_{ST}^{AT} = TFTD_{ST}^{AT} + |t_f^{AT} - t_f^{ST}|$
15: **else if** $(y_1^{ST} - h^{AT} + 1 \leqslant y_1^{AT} \leqslant y_1^{ST}$ **and** $y_2^{ST} - h^{AT} + 1 \leqslant y_1^{AT} \leqslant y_2^{ST}$ **and** $x_1^{AT} = x_2^{ST} + 1)$ **or** $(y_1^{ST} - h^{AT} + 1 \leqslant y_1^{AT} \leqslant y_1^{ST}$ **and** $y_2^{ST} - h^{AT} + 1 \leqslant y_1^{AT} \leqslant y_2^{ST}$ **and** $x_1^{AT} = x_1^{ST} - w^{AT})$ **then**
16: $TCV_{ST}^{AT} = TCV_{ST}^{AT} + (y_2^{ST} - y_1^{ST} + 1) \times min(lt^{AT}, (t_f^{ST} - t_s^{AT}))$
17: $TFTD_{ST}^{AT} = TFTD_{ST}^{AT} + |t_f^{AT} - t_f^{ST}|$
18: **else if** $(x_2^{ST} - w^{AT} + 1 \leqslant x_1^{AT} \leqslant x_2^{ST}$ **and** $y_1^{AT} = y_1^{ST} - h^{AT})$ **or** $(x_2^{ST} - w^{AT} + 1 \leqslant x_1^{AT} \leqslant x_2^{ST}$ **and** $y_1^{AT} = y_2^{ST} + 1)$ **then**
19: $TCV_{ST}^{AT} = TCV_{ST}^{AT} + (x_2^{ST} - x_1^{AT} + 1) \times min(lt^{AT}, (t_f^{ST} - t_s^{AT}))$
20: $TFTD_{ST}^{AT} = TFTD_{ST}^{AT} + |t_f^{AT} - t_f^{ST}|$ 21: **else if** $(x_1^{ST} - w^{AT} + 1 \leqslant x_1^{AT} \leqslant x_1^{ST}$ **and** $y_1^{AT} = y_1^{ST} - h^{AT})$ **or** $(x_1^{ST} - w^{AT} + 1 \leqslant x_1^{AT} \leqslant x_1^{ST}$ **and** $y_1^{AT} = y_2^{ST} + 1)$ **then**
22: $TCV_{ST}^{AT} = TCV_{ST}^{AT} + (x_1^{AT} + w^{AT} - x_1^{ST}) \times min(lt^{AT}, (t_f^{ST} - t_s^{AT}))$
23: $TFTD_{ST}^{AT} = TFTD_{ST}^{AT} + |t_f^{AT} - t_f^{ST}|$
24: **else if** $(y_2^{ST} - h^{AT} + 1 \leqslant y_1^{AT} \leqslant y_2^{ST}$ **and** $x_1^{AT} = x_2^{ST} + 1)$ **or** $(y_1^{ST} - h^{AT} + 1 \leqslant y_1^{AT} \leqslant y_2^{ST}$ **and** $x_1^{AT} = x_1^{ST} - w^{AT})$ **then**
25: $TCV_{ST}^{AT} = TCV_{ST}^{AT} + (y_2^{ST} - y_1^{AT} + 1) \times min(lt^{AT}, (t_f^{ST} - t_s^{AT}))$
26: $TFTD_{ST}^{AT} = TFTD_{ST}^{AT} + |t_f^{AT} - t_f^{ST}|$
27: **else if** $(y_1^{ST} - h^{AT} + 1 \leqslant y_1^{AT} \leqslant y_1^{ST}$ **and** $x_1^{AT} = x_2^{ST} + 1)$ **or** $(y_1^{ST} - h^{AT} + 1 \leqslant y_1^{AT} \leqslant y_1^{ST}$ **and** $x_1^{AT} = x_1^{ST} - w^{AT})$
28: $TCV_{ST}^{AT} = TCV_{ST}^{AT} + (y_1^{AT} + h^{AT} - y_1^{ST}) \times min(lt^{AT}, (t_f^{ST} - t_s^{AT}))$
29: $TFTD_{ST}^{AT} = TFTD_{ST}^{AT} + |t_f^{AT} - t_f^{ST}|$
30: **end if**
31: **else if** $t_f^{ST} = t_s^{AT}$ **or** $t_f^{AT} = t_s^{ST}$ **then**
32: $THV_{ST}^{AT} = THV_{ST}^{AT} + max(min(x_1^{AT} + w^{AT} - 1, x_2^{ST}) - max(x_1^{AT}, x_1^{ST}) + 1, 0) \times max(min(y_1^{AT} + h^{AT} - 1, y_2^{ST}) - max(y_1^{AT}, y_1^{ST}) + 1, 0)$
33: **end if**
34: **end for**

Please cite this article in press as: Marconi T. Online scheduling and placement of hardware tasks with multiple variants on dynamically reconfigurable field-programmable gate arrays. Comput Electr Eng (2013), http://dx.doi.org/10.1016/j.compeleceng.2013.07.004

To complete our discussion, let us compute TCV_{ST}^{AT}, THV_{ST}^{AT}, and $TFTD_{ST}^{AT}$ in Fig. 9(a) and (b). For both figures (a) and (b), the algorithm executes lines 1–3, the initialization. In figure (a), since the arriving task is started right away after the scheduled task (i.e. $t_f^{ST} = t_s^{AT}$), the condition in line 5 cannot be satisfied; the algorithm proceeds to line 31. The algorithm skips lines 6-30; hence, $TCV_{ST}^{AT} = 0$ and $TFTD_{ST}^{AT} = 0$. In line 31, the condition of $t_f^{ST} = t_s^{AT}$ is satisfied; therefore, the total hiding value is updated for figure (a), $THV_{ST}^{AT} = THV_{ST}^{AT} + max(min(x_1^{AT} + w^{AT} - 1, x_2^{ST}) - max(x_1^{AT}, x_1^{ST}) + 1, 0) \times max\left(min\left(y_1^{AT} + h^{AT} - 1, y_2^{ST}\right) - max(y_1^{AT}, y_1^{ST}) + 1, 0\right) = max(x_1^{AT} + w^{AT} - 1 - x_1^{AT} + 1, 0) \times max(y_2^{ST} - y_1^{AT} + 1, 0) = w^{AT} \times (y_2^{ST} - y_1^{AT} + 1)$. Using the same way, in figure (b), since the scheduled task is started right away after the arriving task (i.e. $t_f^{AT} = t_s^{ST}$) and the starting time of scheduled task is more than the starting time of arriving task (i.e. $t_s^{ST} > t_s^{AT}$), the condition in line 5 cannot be satisfied; the algorithm proceeds to line 31. The condition in line 31 is satisfied by $t_f^{AT} = t_s^{ST}$. As a result, the total hiding value is updated for figure (b), $THV_{ST}^{AT} = THV_{ST}^{AT} + max(min(x_1^{AT} + w^{AT} - 1, x_2^{ST}) - max(x_1^{AT}, x_1^{ST}) + 1, 0) \times max\left(min\left(y_1^{AT} + h^{AT} - 1, y_2^{ST}\right) - max(y_1^{AT}, y_1^{ST}) + 1, 0\right) = max(x_2^{ST} - x_1^{ST} + 1, 0) \times max\left(y_1^{AT} + h^{AT} - 1 - y_1^{ST} + 1, 0\right) = (x_2^{ST} - x_1^{ST} + 1) \times (y_1^{AT} + h^{AT} - y_1^{ST})$. Since the algorithm skips lines 6-30 for figure (b); hence, $TCV_{ST}^{AT} = 0$ and $TFTD_{ST}^{AT} = 0$.

The pseudocode of choosing the best position among best potential candidates is depicted in Algorithm 5. The goals of this algorithm are to maximize α^{AT} and minimize $TFTD_{ST}^{AT}$. If both these goals cannot be satisfied at the same time, the heuristic used here is to prioritize maximizing α^{AT} than minimizing $TFTD_{ST}^{AT}$. In line 1, if the checking position produces higher α^{AT} and lower $TFTD_{ST}^{AT}$ than the current chosen position, the checking position is selected as a new current chosen position, replacing the old one (lines 2–3) and updating α^{max} (line 4) and $TFTD_{ST}^{min}$ (line 5). Since the heuristic is to favor maximizing α^{AT} than minimizing $TFTD_{ST}^{AT}$, the algorithm is designed to change the chosen position (lines 7–8) and update its α^{max} (line 9) as far as α^{AT} can be maximized (line 6). In line 10, if both conditions in lines 1 and 6 cannot be satisfied, the only condition that can change the decision is by minimizing $TFTD_{ST}^{AT}$ while preserving α^{max} at its highest value. This heuristic guides the algorithm to choose the position with the highest α^{max} that produces the lowest $TFTD_{ST}^{AT}$.

Algorithm 5. Choosing the best position

1: **if** $\alpha^{AT} > \alpha^{max}$ **and** $TFTD_{ST}^{AT} < TFTD_{ST}^{min}$ **then**
2: $x^{AT} = x$
3: $y^{AT} = y$
4: $\alpha^{max} = \alpha^{AT}$
5: $TFTD_{ST}^{min} = TFTD_{ST}^{AT}$
6: **else if** $(\alpha^{AT} > \alpha^{max})$ **then**
7: $x^{AT} = x$
8: $y^{AT} = y$
9: $\alpha^{max} = \alpha^{AT}$
10: **else if** $\alpha^{AT} = \alpha^{max}$ **and** $TFTD_{ST}^{AT} < TFTD_{ST}^{min}$ **then**
11: $x^{AT} = x$
12: $y^{AT} = y$
13: $TFTD_{ST}^{min} = TFTD_{ST}^{AT}$
14: **end if**

Algorithm 6 shows how the algorithm updates its linked lists (i.e. EL and RL). If the arriving task can be started immediately at its arrival time (i.e. $SST = a^{AT}$), it is added in the execution list EL, sorted in order of increasing finishing times (line 2). The algorithm needs to schedule this incoming task to run it in the future if there is no available free reconfigurable resources for it at its arrival time (line 3) by adding it to the reservation list RL (line 4); the addition is sorted in order of increasing starting times. During the updating (lines 6–11), each element of the EL is checked by comparing its finishing time with the current time for removing it from the list after finishing executed in the FPGA. Tasks in the RL are moved to the EL if they are ready to be started running on the FPGA during this updating.

Table 2
Some examples of implemented hardware tasks (et_i for 100 operations, rt_i at 100 MHz).

Hardware tasks	$w_i(CLBs)$	$h_i(CLBs)$	$et_i(ns)$	$rt_i(ns)$
functionPOWER	14	32	43,183	252,560
adpcm_decode	10	32	770,302	180,400
adpcm_encode	10	32	1,031,213	180,400
FIR	33	32	1,565,980	595,320
mdct_bitreverse	32	64	449,412	1,136,520
mmul	25	64	57,278	892,980

Please cite this article in press as: Marconi T. Online scheduling and placement of hardware tasks with multiple variants on dynamically reconfigurable field-programmable gate arrays. Comput Electr Eng (2013), http://dx.doi.org/10.1016/j.compeleceng.2013.07.004

Algorithm 6. Updating the linked lists

1: **if** $SST = a^{AT}$ **then**
2: add the AT to the EL
3: **else**
4: add the AT to the RL
5: **end if**
6: **if** RL $\neq \emptyset$ **then**
7: update RL
8: **end if**
9: **if** EL $\neq \emptyset$ **then**
10: update EL
11: **end if**

The worst-case time complexity analysis of the PM is presented in Table 1. In which W, H, N_{ET}, N_{RT}, n are the FPGA width and height, the number of executing tasks in the execution list, the number of reserved tasks in the reservation list and the number of hardware versions for each task, respectively.

6. Evaluation

In this section, we discuss the evaluation of the proposed algorithm in detail. We first detail the experimental setup. Then, based on the prepared setup, we present the evaluation of our proposal with synthetic hardware tasks. Finally, to complete the discussion, we discuss the evaluation of the proposal with real hardware tasks.

6.1. Experimental setup

To evaluate the proposed algorithm, we have built a discrete-time simulation framework in C. The framework was compiled and run under Windows XP operating system on Intel(R) Core(TM)2 Quad Central Processing Unit (CPU) Q9550 at 2.83 GHz Personal Computer (PC) with 3GB of Random-Access Memory (RAM). We model a realistic FPGA with 116 columns and 192 rows of reconfigurable units (Virtex-4 XC4VLX200). To better evaluate the algorithm, (1) we modeled realistic random hardware tasks to be executed on a realistic target device; (2) we evaluated the algorithm not only with synthetic hardware tasks but also with real hardware tasks; and (3) since the algorithm needs to make a decision at runtime, we measured the algorithm performance not only in terms of scheduling and placement quality, which is related to its deadline miss ratio, but also in terms of runtime overhead. Since the algorithms are online, the information of each incoming task is unknown until its arrival time. The algorithm is not allowed to access this information at compile time. Every task set consists of 1000 tasks, each of which has two hardware versions with its own life-time and task size. We assume the task width of the slowest hardware implementation is half of its original variant, running two times slower than its original one. All results are presented as an average value from five independent runs of experiments for each task set. The *relative deadline* of an arriving task is defined as $rd^{AT} = d^{AT} - lt^{AT} - a^{AT}$. It is also randomly generated with the first number and the second number shown in the x-axis of the experimental results as the minimum and maximum values, respectively. Furthermore, our scheduling scheme is non-preemptive – once a task is loaded onto the device it runs to completion.

The proposed algorithm is designed for 2D area model. Therefore for fair comparison, we only compare our algorithm with the fastest and the most efficient algorithms that support 2D area model reported in the literature. Since the RPR [24], the Communication-aware [25], the Classified Stuffing [21], the Intelligent Stuffing were designed only for 1D area model, we do not evaluate them in this simulation study. In [20], the Stuffing was shown to outperform the Horizon; hence we do not include the Horizon in comparison to our proposal. The CR algorithm was evaluated to be superior compared to the original 2D Stuffing [20] and the 2D Window-based Stuffing [26] in [27]. For that reason, we do not compare our algorithm to the original 2D Stuffing and the 2D Window-based Stuffing. In [28], the 3DC algorithm was presented, showing its superiority against the CR algorithm. Therefore, for better evaluate the proposed algorithm, we only compare it with the best algorithms, which is the 3DC for this time. To evaluate the proposed algorithm, we have implemented three different algorithms integrated in the framework: the 3DC [28], the PM, the PM without pruning (labeled as *PM_without_Pruning*); all were written in C for easy integration. The last one is conducted for measuring the effect of pruning in speeding up runtime decisions. The evaluation is based on two performance parameters defined in Section 3: the *deadline miss ratio* and the *algorithm execution time*. The former is an indication of actual performance, while the latter is an indication of the speed of the algorithms.

6.2. Evaluation with synthetic hardware tasks

To model realistically the synthetic hardware tasks, we use the realistic hardware tasks to acquire the information of hardware task size range as a reference for our random task set generator. The task widths and heights are randomly gen-

Please cite this article in press as: Marconi T. Online scheduling and placement of hardware tasks with multiple variants on dynamically reconfigurable field-programmable gate arrays. Comput Electr Eng (2013), http://dx.doi.org/10.1016/j.compeleceng.2013.07.004

erated in the range [7...45] reconfigurable units to model hardware tasks between 49 and 2025 reconfigurable units to mimic the results of synthesized hardware units. The life-times are randomly created by computer in [5...100] time units, while the intertask-arrival periods are randomly chosen between one time unit and 150 time units. The number of incoming tasks per arrival is randomly produced in [1...25].

The experimental result with varying synthetic workloads is depicted in Fig. 14. The algorithm performance is measured by the *deadline miss ratio*, which is to be minimized. To represent the algorithm runtime overhead, we measure the *algorithm execution time*. This overhead is equally important to be minimized owing to its runtime decision. The *algorithm execution time* and *deadline miss ratio* are shown with respect to the *relative deadline*. Different relative deadlines reflect in different types of workloads. The shorter the *relative deadline* means the higher the *system load*, and vice versa.

Without pruning, the PM runs 1.6 times slower than the 3DC on average because it needs to choose the best hardware version before scheduling and placement; however, when we turn on its pruning capability, it runs much faster than the 3DC. Although the proposed PM algorithm needs to consider various selectable hardware implementations during its scheduling and placement, it reduces the runtime overhead significantly (and, hence, provides 5.5 times faster runtime decisions on average as shown in the top figure) compared to the 3DC algorithm. Such a substantial speedup can be attributed to the fact that the PM approach reduces the search space effectively by using its *pruning capability*.

By comparing the PM with and without pruning, we measure *the effect of the pruning capability* on speeding up the PM algorithm. The top figure shows that the pruning approach effectively makes the PM algorithm 9 times faster on average in finding solution at runtime. The pruning not only reduces the runtime overhead of the algorithm but also it preserves the quality of the solution as shown in bottom figure; the PM with pruning has the same deadline miss ratio compared to the one without pruning. In short, it speeds up the algorithm without sacrificing its quality.

Under lighter workloads (i.e. longer the relative deadlines), algorithms can easily meet its deadline constraint. As a result, in the bottom figure, the deadline miss ratio decreases as the system load is set to be lighter.

The benefit of multi mode algorithm is clearly shown in the bottom figure. Since the PM algorithm is adaptable in choosing the best hardware variant for running each incoming task to fulfill its runtime constraint, it absolutely outperforms the 3DC under all system loads as shown by its deadline miss ratio reduced by 76% on average.

6.3. Evaluation with real hardware tasks

We utilize the benchmark set in C code from [32] (e.g. Modified Discrete Cosine Transform (MDCT), matrix multiplication, hamming code, sorting, FIR, Adaptive Differential Pulse Code Modulation (ADPCM), etc.). The benchmarks are translated to Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL) performed by the DWARV [33] C-to-VHDL compiler. Subsequently, the resulted VHDL files are synthesized with the Xilinx ISE 8.2.01i_PR_5 tools [34] targetting

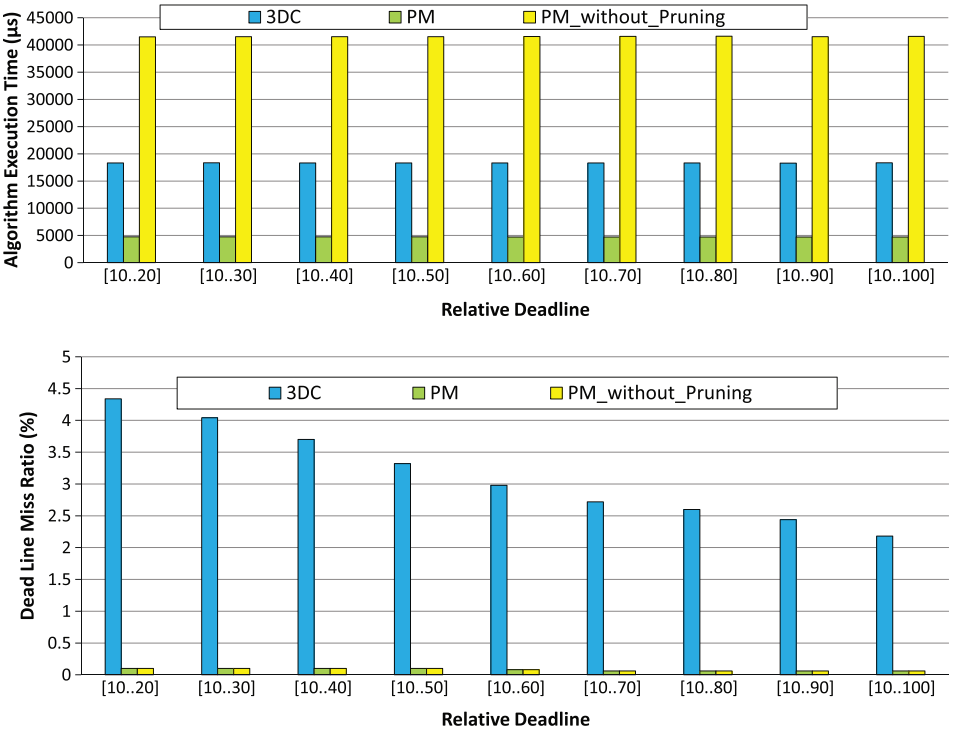

Fig. 15. Evaluation with real hardware tasks.

Please cite this article in press as: Marconi T. Online scheduling and placement of hardware tasks with multiple variants on dynamically reconfigurable field-programmable gate arrays. Comput Electr Eng (2013), http://dx.doi.org/10.1016/j.compeleceng.2013.07.004

Virtex-4 XC4VLX200 device with 116 columns and 192 rows of reconfigurable units. From these hardware implementations, we capture the required resources, the reconfiguration times and the execution times of the hardware tasks. Some examples of implemented hardware tasks are shown in Table 2. For example, the area A_i for function POWER obtained after synthesis is 444 CLBs. In [29], one Virtex-4 row has a height of 16 CLBs. By choosing two rows for implementing this function, we obtain $h_i = 2 \times 16 = 32$ CLBs and $w_i = \lceil A_i/h_i \rceil = \lceil 444/32 \rceil = 14$ CLBs. The function needs 37 cycles with 11.671 ns clock period (85.68 MHz). Hence, we compute the execution time of 100 back-to-back operations to be $et_i = 37 - \times 11.671 \times 100 = 43{,}183$ ns. There are 22 frames per column and each frame contains 1312 bits. Therefore one column needs $22 \times 1312 = 28{,}864$ bits. Since the function occupies 14 CLBs in two rows (32 CLBs), we acquire a bitstream with $14 \times 2 \times 28{,}864 = 808{,}192$ bits. Since ICAP can send 32 bits per clock cycle at 100 MHz, we compute the reconfiguration time $rt_i = 808{,}192 \times 10/32 = 252{,}560$ ns. In the simulation, we assume that the life-time lt_i is the sum of reconfiguration time rt_i and execution time et_i. The hardware tasks are selected randomly from 37 implemented hardware tasks.

Fig. 15 displays the result of evaluating the proposed algorithm with real hardware tasks of varying system loads. Thanks again to its pruning technique the PM makes runtime decisions 4.7 times faster on average than the 3DC in serving each arriving real hardware task under all realistic workloads.

Similar like the experimental result using synthetic hardware tasks, this study with real tasks shows that the multi mode algorithm without pruning runs 1.9 times slower than the single mode one. Another important observation is that the pruning capability effectively tackles this high runtime overhead of multi mode algorithms by limiting the search space while choosing the best solution (top figure); it maintains the same high quality of the scheduling and placement (bottom figure).

A similar performance trend is evident in bottom figure, where the deadline miss ratio is decreased when we do the experimentations with lighter real workloads. Fig. 15 implies that the superiority of our algorithm is not only applicable for synthetic hardware tasks but also for real hardware tasks. Irrespective of the system load, the PM algorithm outperforms the 3DC by more than 90% on average for evaluation with real workloads.

7. Conclusion

In this article, we have discussed an idea of incorporating multiple hardware versions of hardware tasks during task scheduling and placement at runtime targeting partially reconfigurable devices. To better describe the idea, we have presented formal problem definition of online task scheduling and placement with multiple hardware implementations for reconfigurable computing systems. We have shown in this work that by exploring multiple variants of hardware tasks during runtime, on one hand, the algorithms can gain a better performance; however, on the other hand, they can suffer much more runtime overhead in making decision of selecting best implemented hardware on-the-fly. To solve this issue, we have proposed a technique to shortening runtime overhead by reducing search space at runtime. By applying this technique, we have constructed a fast efficient online task scheduling and placement algorithm on dynamically reconfigurable FPGAs incorporating multiple hardware versions, in the sense that it is capable of making fast runtime decisions, without runtime performance loss. The experimental studies show that the proposed algorithm has a much better quality (87% less deadline miss ratio on average) with respect to the state-of-the-art algorithms while providing lower runtime overhead (4.7 times faster on average). Moreover, to relate this work with practical issue of current technology, we have proposed an idea to create an FPGA technology independent environment where different FPGA vendors can provide their consumers with full access to partial reconfigurable resources without exposing all of the proprietary details of the underlying bitstream formats by decoupling area model from the fine-grain details of the physical FPGA fabric. FPGAs with built-in online scheduling and placement hardware including the ICAP controller are worthy to be investigated for future general purpose computing systems.

Acknowledgement

This work has been supported by MOE Singapore research grant MOE2009-T2-1-033.

References

[1] Kao C. Benefits of partial reconfiguration. Xilinx; 2005.
[2] Young SP, Bauer TJ. Architecture and method for partially reconfiguring an FPGA; 2003.
[3] Hussain Hanaa M, Benkrid Khaled, Ebrahim Ali, Erdogan Ahmet T, Seker Huseyin. Novel dynamic partial reconfiguration implementation of *k*-means clustering on FPGAs: comparative results with gpps and gpus. Int J Reconfig Comput 2012;2012.
[4] Shreejith S, Fahmy SA, Lukasiewycz M. Reconfigurable computing in next-generation automotive networks. IEEE Embedded Syst Lett 2013;5(1):12–5.
[5] Niu Xinyu, Jin Qiwei, Luk W, Liu Qiang, Pell O. Exploiting run-time reconfiguration in stencil computation. In: 2012 22nd International conference on field programmable logic and applications (FPL), 2012. p. 173–80.
[6] Jin Qiwei, Becker T, Luk W, Thomas D. Optimising explicit finite difference option pricing for dynamic constant reconfiguration. In: 2012 22nd International conference on field programmable logic and applications (FPL), 2012 p. 165–72.
[7] Devaux L, Pillement Sebastien, Chillet D, Demigny D. R2noc: dynamically reconfigurable routers for flexible networks on chip. In: 2010 International conference on reconfigurable computing and FPGAs (ReConFig), 2010. p. 376–81.
[8] Hussain M, Din A, Violante M, Bona B. An adaptively reconfigurable computing framework for intelligent robotics. In: 2011 IEEE/ASME international conference on advanced intelligent mechatronics (AIM), 2011. p. 996–1002.
[9] He Ke, Crockett Louise, Stewart Robert. Dynamic reconfiguration technologies based on FPGA in software defined radio system. J Signal Process Syst 2012;69(1):75–85.

Please cite this article in press as: Marconi T. Online scheduling and placement of hardware tasks with multiple variants on dynamically reconfigurable field-programmable gate arrays. Comput Electr Eng (2013), http://dx.doi.org/10.1016/j.compeleceng.2013.07.004

[10] Crowe F, Daly A, Kerins T, Marnane W. Single-chip FPGA implementation of a cryptographic co-processor. In: Proceedings of the 2004 IEEE international conference on field-programmable technology, 2004. December 2004. p. 279–85.
[11] Shayee KRS, Park Joonseok, Diniz PC. Performance and area modeling of complete FPGA designs in the presence of loop transformations. In: FCCM 2003. 11th Annual IEEE symposium on field-programmable custom computing machines, 2003; April 2003. p. 296.
[12] Buyukkurt Betul, Guo Zhi, Najjar Walid A. Impact of loop unrolling on area, throughput and clock frequency in roccc: C to vhdl compiler for FPGAs. In: ARC, 2006. p. 401–12.
[13] Dragomir Ozana Silvia, Stefanov Todor, Bertels Koen. Optimal loop unrolling and shifting for reconfigurable architectures. ACM Trans Reconfig Technol Syst 2009;2(4):25:1–25:24.
[14] Nawaz Z, Marconi T, Bertels K, Stefanov T. Flexible pipelining design for recursive variable expansion. In: IPDPS 2009. IEEE international symposium on parallel distributed processing, 2009; May 2009. p. 1–8.
[15] Govindu G, Zhuo L, Choi S, Gundala P, Prasanna V. Area and power performance analysis of a floating-point based application on FPGAs. 2003.
[16] Nibouche O, Belatreche A, Nibouche M. Speed and area trade-offs for FPGA-based implementation of rsa architectures. In: NEWCAS 2004. The 2nd annual IEEE northeast workshop on circuits and systems, 2004; June 2004. p. 217–20.
[17] Homsirikamol Ekawat, Rogawski Marcin, Gaj Kris. Throughput vs. area trade-offs in high-speed architectures of five round 3 sha-3 candidates implemented using xilinx and altera FPGAs. In: CHES'11. Proceedings of the 13th international conference on cryptographic hardware and embedded systems. Verlag (Berlin, Heidelberg): Springer; 2011. p. 491–506.
[18] Sutter G, Boemo E. Experiments in low power FPGA design. Latin Am Appl Res 2007;37:99–104.
[19] Munoz DM, Sanchez DF, Llanos CH, Ayala-Rincon M. Tradeoff of FPGA design of floating-point transcendental functions. In: 2009 17th IFIP international conference on very large scale integration (VLSI-SoC); October 2009. p. 239–42.
[20] Steiger Christoph, Walder Herbert, Platzner Marco. Operating systems for reconfigurable embedded platforms: online scheduling of real-time tasks. IEEE Trans Comput 2004;53(11):1393–407.
[21] Hsiung Pao-Ann, Huang Chun-Hsian, Chen Yuan-Hsiu. Hardware task scheduling and placement in operating systems for dynamically reconfigurable SoC. J Embedded Comput 2009;3(1):53–62.
[22] Marconi Thomas, Lu Yi, Bertels Koen, Gaydadjiev Georgi. Online hardware task scheduling and placement algorithm on partially reconfigurable devices. In: ARC, 2008. p. 302–7.
[23] Marconi Thomas. Efficient runtime management of reconfigurable hardware resources. PhD thesis, Computer Engineering Lab, TU Delft; June 2011.
[24] Lu Yi, Marconi Thomas, Bertels Koen, Gaydadjiev Georgi. Online task scheduling for the FPGA-based partially reconfigurable systems. In: ARC, 2009. p. 216–30.
[25] Lu Yi, Marconi T, Bertels K, Gaydadjiev G. A communication aware online task scheduling algorithm for FPGA-based partially reconfigurable systems. In: 2010 18th IEEE annual international symposium on field-programmable custom computing machines (FCCM), 2010. p. 65–8.
[26] Zhou Xue-Gong, Wang Ying, Huang Xun-Zhang, Peng Cheng-Lian. On-line scheduling of real-time tasks for reconfigurable computing system. In: International conference on field programmable technology (FPT), 2006. p. 57–64.
[27] Zhou Xuegong, Wang Ying, Huang XunZhang, Peng Chenglian. Fast on-line task placement and scheduling on reconfigurable devices. In: FPL, 2007. p. 132–8.
[28] Marconi Thomas, Lu Yi, Bertels Koen, Gaydadjiev Georgi. 3d compaction: a novel blocking-aware algorithm for online hardware task scheduling and placement on 2d partially reconfigurable devices. In: ARC, 2010. p. 194–206.
[29] Xilinx. Virtex-4 FPGA configuration user guide, Xilinx user guide UG071. Xilinx, Inc., 2008.
[30] Vassiliadis S, Wong S, Gaydadjiev G, Bertels K, Kuzmanov G, Panainte EM. The MOLEN polymorphic processor. IEEE Trans Comput 2004;53(11):1363–75.
[31] Rossmeissl C, Sreeramareddy A, Akoglu A. Partial bitstream 2-d core relocation for reconfigurable architectures. In: AHS 2009. NASA/ESA conference on adaptive hardware and systems, 2009. 29, 2009–August 1, 2009. p. 98–105.
[32] Meeuws R, Yankova Y, Bertels K, Gaydadjiev G, Vassiliadis S. A quantitative prediction model for hardware/software partitioning. In: FPL 2007. international conference on field programmable logic and applications, 2007. August 2007. p. 735–9.
[33] Y. Yankova, K. Bertels, S. Vassiliadis, R. Meeuws, A. Virginia, Automated HDL generation: comparative evaluation. In: ISCAS 2007. IEEE international symposium on circuits and systems, 2007. May 2007. p. 2750–3.
[34] Xilinx. Early access partial reconfiguration user guide, Xilinx user guide UG208. Xilinx, Inc.; 2006.

Thomas Marconi received his PhD degree in Reconfigurable Computing from TU Delft, in 2011 after working on hArtes project. From 2010 to 2013, he was with NUS, Singapore, where he worked on bahurupi project. Currently, he is with TU Delft, working on i-RISC project. His current research interests include coding theory, reconfigurable computing, embedded systems. He is an affiliate member of European Network of Excellence HiPEAC.

Please cite this article in press as: Marconi T. Online scheduling and placement of hardware tasks with multiple variants on dynamically reconfigurable field-programmable gate arrays. Comput Electr Eng (2013), http://dx.doi.org/10.1016/j.compeleceng.2013.07.004