A Novel Online Hardware Task Scheduling and Placement Algorithm for 3D Partially Reconfigurable FPGAs

Thomas Marconi, Tulika Mitra
School of Computing
National University of Singapore
{marconi,tulika}@comp.nus.edu.sg

***Abstract*—The recent emergence of 3D partially reconfigurable FPGAs implies that we need efficient online hardware task scheduling and placement algorithms for such architectures. However, the algorithms available in the literature for 3D FPGAs create a "blocking-effect". That is, these algorithms tend to make a wrong decision in finding a location of each arriving hardware task during runtime scheduling and placement on 3D partially reconfigurable FPGAs. This leads to currently scheduled tasks blocking future hardware tasks from being scheduled and satisfying their deadlines. We need to solve this problem to maximize the performance of partially reconfigurable runtime systems implemented using 3D chip technology. We propose a novel placement and scheduling algorithm with a blocking-aware heuristic to make better decisions at runtime. Based on evaluation using both synthetic and real workloads, our algorithm reduces deadline miss rate by 61% with 15% longer runtime overhead compared to state-of-the-art algorithms.**

I. Introduction and Related Work

Recently we are witnessing the emergence of 3D FPGAs. Some of benefits of exploring 3D FPGAs compared to 2D FPGAs are reductions in wire-length[1], delay[1][2][3], channel width[2], power dissipation[2][3], energy consumption[1], and an increase in logic density[3]. However, online task scheduling and placement algorithms for 3D FPGAs have not been well explored in literature. The only algorithm we are aware of targeting 3D FPGAs is presented in [4]; whereas other existing work only target 2D FPGAs (e.g. [5], [6], [7]).

The algorithms presented in [4] perform compaction in spatial and temporal domain. However, these algorithms do not have so called blocking awareness. That is, these algorithms schedule and place hardware tasks in a way that the currently placed tasks may block future tasks to be scheduled. As a result, many tasks miss their deadlines. We call this issue as "blocking-effect". To solve this issue, we propose an efficient algorithm with an awareness to avoid this effect.

The main contributions of this paper are:

- The first efficient blocking-aware online hardware task scheduling and placement algorithm targeting 3D partially reconfigurable FPGAs;
- An extensive evaluation using both synthetic and real hardware tasks implemented on 3D FPGA.

978-1-4577-1740-6/11/$26.00 **© 2011 IEEE**

The rest of the paper is organized as follows. In Section II, we introduce the problem of online task scheduling and placement targeting 3D partially reconfigurable FPGAs. Our proposed algorithm is presented in Section III. The algorithm is evaluated in Section IV. Finally, we conclude in Section V.

II. Problem Definition

A 3D FPGA, denoted by FPGA (W, H, TH) contains $W \times H \times TH$ reconfigurable hardware units arranged in a 3D 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), j^{th} position in the second coordinate (y) and k^{th} position in the third coordinate (z) is identified by coordinate (i, j, k), counted from the lower-leftmost coordinate *(1,1,1)*, where $1 \leq i \leq W$, $1 \leq j \leq H$, and $1 \leq k \leq TH$.

A hardware task is denoted by $T(a, w, h, th, lt, d)$. The task arrives to the system at time a and requires a region of size $w \times h \times th$ in the 3D FPGA (W, H, TH) during its lifetime lt. We define $lt = rt + et$ where rt is the reconfiguration time and et is the execution time of the task. w, h, and th are the task width, height, and thickness respectively where $1 \leq w \leq W$, $1 \leq h \leq H$, $1 \leq th \leq TH$. d is the deadline of the task.

Online task scheduling and placement algorithms targeting 3D FPGAs have to find a region of hardware resources inside the FPGA for running 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 and upper-rightmost corners (x_1, y_1, z_1) and (x_2, y_2, z_2), respectively) for executing the task in the future. The running or scheduled task is denoted as $T(x_1, y_1, z_1, x_2, y_2, z_2, t_s, t_f)$ where $t_f = t_s + lt$ is the finishing time of the task. The hardware task meets its deadline if $t_f \leq d$. We 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 define 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 the time needed to schedule and place the arriving task.

III. Proposed Algorithm

A. Motivational Example

We first present a simple example to explain the idea behind our proposed blocking-aware algorithm. Let us assume that we have a task set as shown in Table I. Blocking-unaware algorithms do not have the ability to avoid choosing placements for current arriving tasks that will become obstacles for future arriving tasks to be scheduled earlier. As a result, the future arriving tasks will miss their deadlines. In this simple example, task T_3 prevents task T_4 to be scheduled earlier. The penalty for this wrong placement decision of T_3 is that the task T_4 misses its deadline as shown in Figure 1(a).

To solve this problem, we introduce an algorithm that has awareness to avoid placements that will be obstacles for future tasks. This can be done for this example by placing task T_3 to a different location as shown in Figure 1(b). Because of this better decision, the blocking-aware algorithm can avoid task T_3 from being an encumbrance for task T_4 to be started earlier. By scheduling T_4 earlier, task T_4 now can finish execution earlier to satisfy its deadline constraint.

To equip the algorithm with the necessary knowledge to avoid the "blocking-effect", the algorithm places an arriving task at a location that hides it inside the previously scheduled tasks as much as possible. To discuss this idea more concretely, let us introduce some definitions here. A task T_i is called a *previous scheduled task* PST (*next scheduled task* NST) of task T_j if task T_j (T_i) starts execution right after task T_i (T_j) finishes. In this example, task T_2 is a previous scheduled task of task T_3. The idea behind our proposed algorithm is to choose a position for an arriving task that overlaps as much as possible with its PSTs and NSTs. We quantify this overlap as *hiding value* later in our discussion. In this example, we can see that the placement for task T_3 in Figure 1(b) has a higher *hiding value* than its placement in Figure 1(a).

TABLE I
An example of task set

T_i	a_i	w_i	h_i	th_i	lt_i	d_i
T_1	1	6	10	10	10	12
T_2	1	4	10	10	20	24
T_3	2	8	10	10	10	32
T_4	3	2	10	10	25	40

B. Acceptable Region inside FPGA

To equip our algorithm with an ability to avoid placing tasks outside the FPGA, we give it an awareness of *acceptable region*. The acceptable region inside FPGA (W, H, TH) with respect to an arriving hardware task $AT(a, w, h, th, lt, d)$ is a region where the algorithm can place the *origin* of the arriving task AT without exceeding the FPGA boundary. The acceptable region has the lower-leftmost and upper-rightmost

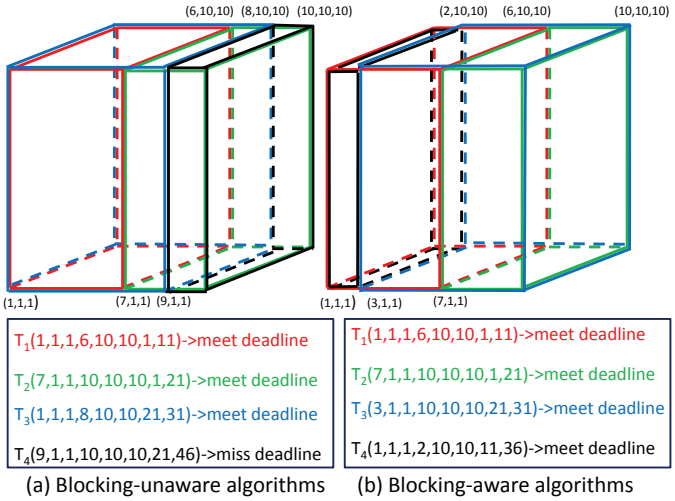

Fig. 1. Scheduling and placement of task set in Table I by blocking-unaware algorithm versus blocking-aware algorithm.

corners of $(1, 1, 1)$ and $(W - w + 1, H - h + 1, TH - th + 1)$, respectively. This information is also needed by the algorithm to limit its search region in finding the best position for each arriving task so as to lower the runtime overhead.

C. Conflicting Region with Scheduled Tasks

To avoid our algorithm placing tasks that can conflict with other tasks, we give it an awareness of *conflicting region*. The conflicting region of an arriving hardware task $AT(a, w, h, th, lt, d)$ with respect to a scheduled task $ST(x_1, y_1, z_1, x_2, y_2, z_2, t_s, t_f)$ 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 and upper-rightmost corners at $(max(1, x_1 - w + 1), max(1, y_1 - h + 1), max(1, z_1 - th + 1))$ and $(min(x_2, W - w + 1), min(y_2, H - h + 1), min(z_2, TH - th + 1))$, respectively. The algorithm exploits this information to lower the runtime overhead further.

D. Compaction Value

We attempt to reserve as much free region as possible in the middle of the FPGA to better accommodate future tasks. Hence the algorithm spreads hardware tasks close to the FPGA boundary. To quantize this choice, we introduce *compaction value* with FPGA boundary (CV_{FPGA}) as illustrated in Figure 2. In this simple example, the arriving task AT at position (1,1,1) has three common surfaces with FPGA boundary, i.e., left, bottom, and front surfaces. Therefore, the compaction value with FPGA boundary is the sum of these common surfaces which can be formulated as $CV_{FPGA} = (w \times h + w \times th + h \times th) \times lt$. As there are a lot of positions where the arriving task AT can be placed in the FPGA, we need to provide a general formula as formulated in Table II.

To place each arriving task close to other scheduled tasks both in three-dimensional coordinates (compaction in 3D

TABLE II
COMPUTATIONS OF COMPACTION VALUE WITH FPGA BOUNDARY (CV_{FPGA})

Conditions	CV_{FPGA}
$x = 1, y = 1, z = 1$	$(w * h + w * th + h * th) * lt$
$x = 1, y + h - 1 = H, z = 1$	$(w * h + w * th + h * th) * lt$
$x + w - 1 = W, y + h - 1 = H, z = 1$	$(w * h + w * th + h * th) * lt$
$x + w - 1 = W, y = 1, z = 1$	$(w * h + w * th + h * th) * lt$
$x = 1, y > 1, y + h - 1 < H, z = 1$	$(w * h + h * th) * lt$
$x > 1, x + w - 1 < W, y + h - 1 = H, z = 1$	$(w * h + w * th) * lt$
$x + w - 1 = W, y > 1, y + h - 1 < H, z = 1$	$(w * h + h * th) * lt$
$x > 1, x + w - 1 < W, y = 1, z = 1$	$(w * h + w * th) * lt$
$x > 1, x + w - 1 < W, y > 1, y + h - 1 < H, z = 1$	$(w * h) * lt$
$x = 1, y = 1, z + th - 1 = TH$	$(w * h + w * th + h * th) * lt$
$x = 1, y + h - 1 = H, z + th - 1 = TH$	$(w * h + w * th + h * th) * lt$
$x + w - 1 = W, y + h - 1 = H, z + th - 1 = TH$	$(w * h + w * th + h * th) * lt$
$x + w - 1 = W, y = 1, z + th - 1 = TH$	$(w * h + w * th + h * th) * lt$
$x = 1, y > 1, y + h - 1 < H, z + th - 1 = TH$	$(w * h + h * th) * lt$
$x > 1, x + w - 1 < W, y + h - 1 = H, z + th - 1 = TH$	$(w * h + w * th) * lt$
$x + w - 1 = W, y > 1, y + h - 1 < H, z + th - 1 = TH$	$(w * h + h * th) * lt$
$x > 1, x + w - 1 < W, y = 1, z + th - 1 = TH$	$(w * h + w * th) * lt$
$x > 1, x + w - 1 < W, y > 1, y + h - 1 < H, z + th - 1 = TH$	$(w * h) * lt$
$x = 1, y = 1, z > 1, z + th - 1 < TH$	$(w * th + h * th) * lt$
$x = 1, y > 1, y + h - 1 < H, z > 1, z + th - 1 < TH$	$(h * th) * lt$
$x = 1, y + h - 1 = H, z > 1, z + th - 1 < TH$	$(w * th + h * th) * lt$
$x + w - 1 = W, y = 1, z > 1, z + th - 1 < TH$	$(w * th + h * th) * lt$
$x + w - 1 = W, y > 1, y + h - 1 < H, z > 1, z + th - 1 < TH$	$(h * th) * lt$
$x + w - 1 = W, y + h - 1) = H, z > 1, z + th - 1 < TH$	$(w * th + h * th) * lt$
$x > 1, x + w - 1 < W, y = 1, z > 1, z + th - 1 < TH$	$(w * th) * lt$
$x > 1, x + w - 1 < W, y + h - 1 = H, z > 1, z + th - 1 < TH$	$(w * th) * lt$

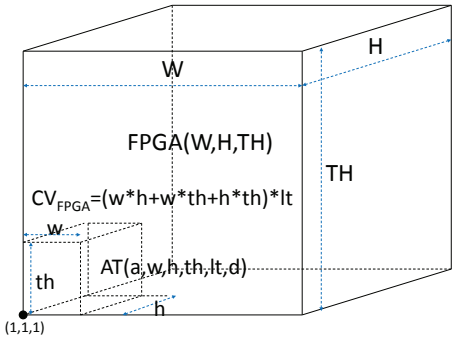

Fig. 2. An example of compaction value with FPGA(W, H, TH) boundary for an arriving task $AT(a, w, h, th, lt, d)$ at position (1,1,1)

space) and in time coordinate (compaction in time), in addition to the compaction value with FPGA boundary, the proposed algorithm also computes a quantity called *compaction value* with scheduled tasks (CV_{ST}). A simple example of how to compute this value is shown in Figure 3. In this example, the overlapped area between the bottom side of the arriving task AT and the top side of the scheduled task ST is the compaction value with respect to that scheduled task. As a result, CV_{ST} can be computed in this case as $CV_{ST} = (x + w - x_1) \times (y + h - y_1) \times min(lt, ((t_f - t_s))$. As our algorithm needs to compact tasks not only in the three-dimensional domain but also in the time domain, the term $min(lt, ((t_f - t_s))$ is added in this computation. The placement position of AT can be in any free region of FPGA. So there are a number of ways in which AT can overlap with ST and we need a general formula to compute those values as presented in Table III. The sum of the compaction value with FPGA boundary and the compaction value with scheduled tasks is called as *total compaction value* and is formulated as $CV = CV_{FPGA} + CV_{ST}$ (1). This value guides our algorithm to place tasks as compactly as possible in four dimensions (x, y, z, and time coordinates).

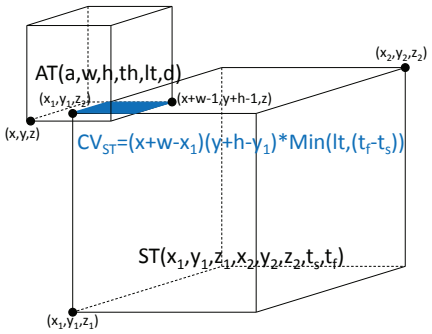

Fig. 3. An example of compaction value for an arriving task $AT(a, w, h, th, lt, d)$ at position $z = z_2 + 1$ with respect to scheduled task $ST(x_1, y_1, z_1, x_2, y_2, z_2, t_s, t_f)$

E. Hiding Value

In addition to the above compaction, the proposed algorithm is also instrumented with an ability to maximize the *hiding value* as mentioned previously. The simple example of how to compute the hiding value is illustrated in Figure 4. In this figure, the volume of the region with the upper-leftmost corner (x_1, y_1, z_2) and the lower-rightmost corner $(x+w-1, y+h-1, z)$ is the hiding value that we are looking for. Therefore, the HV for this position can be formulated as $HV = (x + w - x_1) \times (y + h - y_1) \times (z_2 - z + 1)$. We introduce the general formula for computing HV for any possible condition as $HV = max(min(x + w - 1, x_2) - max(x, x_1) + 1, 0) \times$

TABLE III
COMPUTATIONS OF COMPACTION VALUE WITH SCHEDULED TASK(CV_{ST})

Conditions	CV_{ST}
$z = z_2 + 1$	$Max(Min(x+w-1, x_2) - Max(x, x_1) + 1, 0) * Max(Min(y+h-1, y_2) - Max(y, y1) + 1, 0) * Min(lt, (t_f - t_s))$
$z_1 = z + th$	$Max(Min(x+w-1, x_2) - Max(x, x_1) + 1, 0) * Max(Min(y+h-1, y_2) - Max(y, y1) + 1, 0) * Min(lt, (t_f - t_s))$
$y = y_2 + 1$	$Max(Min(x+w-1, x_2) - Max(x, x_1) + 1, 0) * Max(Min(z+th-1, z_2) - Max(z, z1) + 1, 0) * Min(lt, (t_f - t_s))$
$y_1 = y + h$	$Max(Min(x+w-1, x_2) - Max(x, x_1) + 1, 0) * Max(Min(z+th-1, z_2) - Max(z, z1) + 1, 0) * Min(lt, (t_f - t_s))$
$x = x_2 + 1$	$Max(Min(y+h-1, y_2) - Max(y, y_1) + 1, 0) * Max(Min(z+th-1, z_2) - Max(z, z1) + 1, 0) * Min(lt, (t_f - t_s))$
$x_1 = x + w$	$Max(Min(y+h-1, y_2) - Max(y, y_1) + 1, 0) * Max(Min(z+th-1, z_2) - Max(z, z1) + 1, 0) * Min(lt, (t_f - t_s))$

$max(min(y+h-1, y_2) - max(y, y1) + 1, 0) \times max(min(z+th-1, z_2) - max(z, z_1) + 1, 0)$ (2).

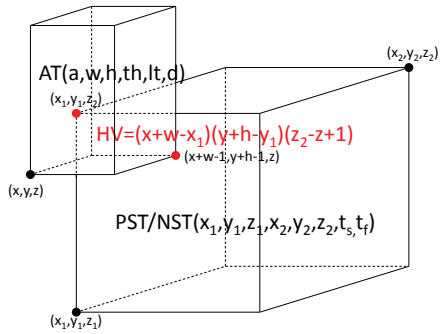

Fig. 4. An example of hiding value with previous or next scheduled task $PST/NST(x_1, y_1, z_1, x_2, y_2, z_2, t_s, t_f)$ for an arriving task $AT(a, w, h, th, lt, d)$

F. Pseudocode and Analysis

The pseudocode of our blocking-aware algorithm, called *4D Compaction (4DC)*, is shown in Algorithm 1. Two linked lists (the execution list (EL) and the reservation list (RL)) are maintained. The EL records the information of all currently running hardware tasks sorted in order of increasing finish times; whereas the RL contains the information of all scheduled tasks sorted in order of increasing starting times. The information stored in the lists are the lower-leftmost corner coordinate (x_1, y_1, z_1), the upper-rightmost corner coordinate (x_2, y_2, z_2), the starting time t_s, the finishing time t_f, the task name, the next pointer, and the previous pointer.

In lines 1-29, the algorithm computes the 3D starting time matrix $STM(x, y, z)$ for the arriving task volume $w \times h \times th$ inside the FPGA volume $W \times H \times TH$. This matrix records the earliest starting time for each potential position of the corresponding arriving task. The algorithm collects all possible positions that have enough space for the arriving task by scanning the EL and RL. The algorithm fills each element of the STM with the arrival time of incoming task a (lines 1-7). As shown, the algorithm only needs to create the STM matrix for *acceptable region* as presented previously to minimize runtime overhead. The algorithm updates groups of elements that are affected by all executing tasks in the EL (lines 8-18) and by all scheduled tasks in RL (lines 19-29). The algorithm only needs to update the affected elements limited by the *conflicting region* as defined before to reduce runtime overhead further.

In line 30, the algorithm collects all best positions (candidates) that have the earliest starting time (best starting time positions: best positions in terms of starting time) from the STM. Since the algorithm is not only designed to choose the best position for each arriving task in terms of starting time (time domain) but also the best position in terms of space (space domain), it needs to pack tasks compactly further. The algorithm computes the compact value CV (line 32) using formulas in Table II, Table III, and equation (1) and the algorithm chooses the best position from all the best starting time positions. For reducing runtime overhead further, the algorithm does not need to compute the compaction value for all positions, it only computes the compaction value for the best positions (candidates) (line 31). Intuitively, the highest compaction value gives the best position in terms of packing tasks in four dimensions as shown previously.

Besides the compaction value, the algorithm also uses the sum of finish time difference (SFTD) heuristic for all scheduled tasks that contacted in three-dimensional space with the arriving task (referred as VC set). The algorithm computes current SFTD ($c\_SFTD = \sum_{\forall tasks \in VC} |t_s + lt - t_f|$) in line 33. The SFTD heuristic gives our algorithm an ability to group tasks with similar finish times to get large free volume during de-allocations. Moreover, to avoid "blocking-effect", it computes the hiding value in line 34 using equation (2).

The algorithm chooses the position with the highest compaction value, the lowest SFTD value, and the highest hiding value for allocating the arriving task (lines 35-57). Allocating the arriving tasks at the highest compaction value compacts the tasks both in time and space; while grouping tasks with similar finish times creates more possibilities to produce larger free space during de-allocations. On top of that, the highest hiding value guides our algorithm to avoid "blocking-effect".

The algorithm allocates the arriving task when space is available for the task; otherwise, the algorithm needs to schedule the task for future execution. If the arriving task can be allocated at its arrival time (line 59), it will be executed immediately and added in the EL (line 60); otherwise, it is inserted in the RL (line 62).

When the tasks in the RL are executed, they are removed from the RL and added in the EL. The finished tasks in the EL are deleted after execution. These updating processes are executed when the lists are not empty (lines 64-69).

The time complexity analysis of our proposed algorithm is presented in Table IV where W, H, TH, N_{ET}, N_{RT} are the FPGA width, the FPGA height, the FPGA thickness, the

Algorithm 1 4D Compaction Algorithm

```
1: for (x=1;x≤W-w+1;x++) do
2:     for (y=1;x≤H-h+1;y++) do
3:         for (z=1;x≤TH-th+1;z++) do
4:             STM(x,y,z)=a
5:         end for
6:     end for
7: end for
8: for all tasks in the EL do
9:     for (x=max(1,x1-w+1);x≤min(x2,W-w+1);x++) do
10:        for (y=max(1,y1-h+1);y≤min(y2,H-h+1);y++) do
11:            for (z=max(1,z1-th+1);z≤min(z2,TH-th+1);z++) do
12:                if STM(x,y,z)<tf then
13:                    STM(x,y,z)=tf
14:                end if
15:            end for
16:        end for
17:    end for
18: end for
19: for all tasks in the RL do
20:    for (x=max(1,x1-w+1);x≤min(x2,W-w+1);x++) do
21:        for (y=max(1,y1-h+1);y≤min(y2,H-h+1);y++) do
22:            for (z=max(1,z1-th+1);z≤min(z2,TH-th+1);z++) do
23:                if (STM(x,y,z)<tf and STM(x,y,z)+lt>ts) then
24:                    STM(x,y,z)=tf
25:                end if
26:            end for
27:        end for
28:    end for
29: end for
30: collect all positions from STM that have the earliest starting time
31: for all above positions do
32:    c_CV=compute compact value
33:    s_SFTD=compute sum of finishing time difference
34:    s_HV=compute hiding value
35:    if (c_CV>CV_max and c_SFTD<SFTD_min and c_HV>HV_max) then
36:        best position=current position
37:        CV_max=c_CV
38:        SFTD_min=c_SFTD
39:        HV_max=c_HV
40:    else if (c_CV>CV_max and c_HV>HV_max) then
41:        best position=current position
42:        CV_max=c_CV
43:        HV_max=c_HV
44:    else if (c_CV>CV_max) then
45:        best position=current position
46:        CV_max=c_CV
47:    else if (c_CV=CV_max and c_HV>HV_max) then
48:        best position=current position
49:        HV_max=c_HV
50:    else if (c_CV>CV_max and c_SFTD<SFTD_min) then
51:        best position=current position
52:        CV_max=c_CV
53:        SFTD_min=c_SFTD
54:    else if (c_CV=CV_max and c_SFTD<SFTD_min) then
55:        best position=current position
56:        SFTD_min=c_SFTD
57:    end if
58: end for
59: if best starting time=arrival time then
60:    add the arriving task to the EL
61: else
62:    add the arriving task to the RL
63: end if
64: if RL≠∅ then
65:    update RL
66: end if
67: if EL≠∅ then
68:    update EL
69: end if
```

number of executing tasks in the EL, the number of reserved tasks in the RL, respectively.

IV. Evaluation

A. Evaluation with synthetic hardware tasks

We have built a discrete-time simulation framework in C to evaluate our algorithm. The framework was compiled and run under Windows XP operating system on Intel(R) Core(TM)2 Quad CPU Q9550 at 2.83 GHz PC with 3GB of RAM.

TABLE IV
Time complexity analysis of 4D Compaction algorithm

Line(s)	Time Complexity
1-7	$O(W*H*TH)$
8-18	$O(W*H*TH*N_{ET})$
19-29	$O(W*H*TH*N_{RT})$
30	$O(W*H*TH)$
31-58	$O(W*H*TH*max(N_{ET},N_{RT}))$
59-63	$O(max(N_{ET},N_{RT}))$
64-66	$O(N_{RT})$
67-69	$O(N_{ET})$
Total	$O(W*H*TH*max(N_{ET},N_{RT}))$

We generated 500 random tasks for each task set. Every hardware task has its arriving time, size (width, height, thickness), life-time and deadline. The task widths, heights and thicknesses are randomly generated in the range [5..30] reconfigurable units. The life-times are also randomly generated in [5..100] time units, while the inter-task arrival periods are randomly chosen between one time unit and 50 time units. Total tasks per arrival are randomly generated between [1..5]. Since the algorithms are online, the information about arriving tasks are unknown until their arrival times. The algorithm is not allowed to access this information at compile time. We model a 3D FPGA with 50x50x50 reconfigurable units.

Our algorithm is designed for 3D FPGA. For fair comparison, we only compare our algorithm with existing algorithms that target 3D FPGAs. Based on our literature survey, there are two heuristics used in [4] that focus on 3D FPGAs. These heuristics are called *3D adjacency* and *4D adjacency*.

To evaluate the proposed algorithm, we have implemented three different algorithm:

- *3D adjacency heuristic* (3D_Adj) algorithm [4];
- *4D adjacency heuristic* (4D_Adj) algorithm [4];
- our algorithm using blocking-awareness heuristic, called *4D Compaction* (4DC) algorithm.

The evaluation is based on two performance parameters defined in Section II: the *deadline miss ratio* and the *algorithm execution time*. The experimental results with synthetic hardware tasks are presented in Figure 5. All results are presented as an average value from 10 runs of experiments for each task set. The *relative deadline* of a hardware task in this figure is defined as $rd = d - lt - a$. It is also randomly generated with the first number and the second number shown in the figure as the minimum and maximum values, respectively. The shorter relative deadline makes it more difficult for scheduling and placement algorithm to meet task deadlines. As a result, the deadline miss ratio increases as the relative deadline decreases.

The 3D_Adj has the highest deadline miss ratio due to its spatial-only compaction. Since the 4D_Adj performs additional compaction in time domain on top of its 3D spatial domain, it has better scheduling and placement quality compared to the algorithm using 3D adjacency heuristic, i.e, it has up to 21% lower deadline miss ratio than the 3D_Adj. This figure also shows that our algorithm using blocking-awareness heuristic has the lowest deadline miss ratio, i.e, up to 26% and

11% lower deadline miss ratio compared to the 3D_Adj and 4D_Adj heuristics, respectively. This is due to the fact that our proposed algorithm is not only doing four-dimensional packing of hardware tasks, but it also has an ability to avoid "blocking-effect" as presented previously. By avoiding this effect, tasks can be scheduled earlier to meet their deadlines.

As the 3D_Adj does not need to care about compaction in time domain, it has the lowest runtime overhead. 4D_Adj is aware of time domain and it has 14% higher runtime overhead to choose the best position for each arriving task. Based on the fact that the algorithm using blocking-aware heuristic needs to avoid "blocking-effect", it requires 12% longer time to choose the best solution for arriving tasks compared to 4D_Adj.

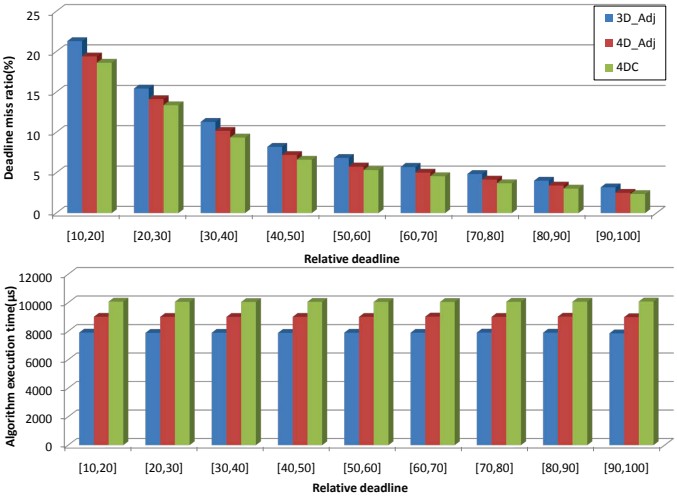

Fig. 5. Evaluation with synthetic hardware tasks

B. Evaluation with real hardware tasks

To complete the evaluation, the algorithm is also evaluated using real 3D hardware tasks. The 3D FPGA implementation of MCNC [8] benchmark circuits obtained from [9] are used for this purpose. The results of the experiments are shown in Figure 6. The figure shows that the superiority of our algorithm is not only applicable for synthetic tasks but also for real tasks. Evaluation with real tasks shows that our algorithm has 61% and 23% lower deadline miss ratio on an average compared to the algorithms using 3D and 4D adjacency heuristics, respectively. The proposed algorithm needs 15% longer runtime overhead to incorporate blocking awareness.

V. Conclusion

In this paper, we have introduced "blocking-effect" in online scheduling and placement of tasks on 3D partially reconfigurable FPGAs. To solve this issue, we propose a blocking-aware heuristic. The proposed heuristic is used to build a novel placement and scheduling algorithm supporting blocking-awareness, named as 4D Compaction. Because of its 4D compaction capability, the proposed algorithm places or schedules the arriving tasks more compactly on a 3D partially

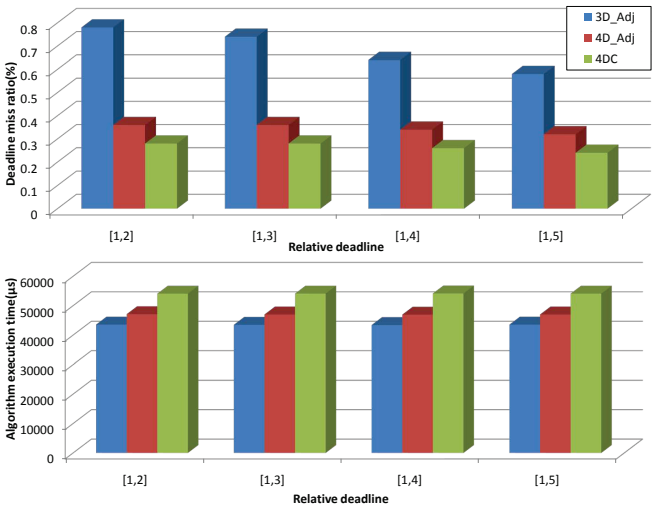

Fig. 6. Evaluation with real hardware tasks

reconfigurable FPGA by doing four-dimensional compaction, i.e, both in 3D spatial coordinates and time coordinate. Moreover, the algorithm is equipped with an ability to group tasks with similar finish times to form larger free volume for better allocation of future tasks. Finally, the algorithm is armed with an ability to avoid "blocking-effect". The algorithm is evaluated using both synthetic and real workloads. Based on this evaluation, the proposed algorithm produces up to 61% better solutions with 15% longer runtime overhead compared to the state-of-the-art schemes.

Acknowledgment

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

References

[1] K. Siozios, K. Sotiriadis, V. F. Pavlidis, and D. Soudris, "Exploring alternative 3d fpga architectures: Design methodology and cad tool support." in *FPL'07*, 2007, pp. 652–655.

[2] A. Rahman, S. Das, A. P. Chandrakasan, and R. Reif, "Wiring requirement and three-dimensional integration technology for field programmable gate arrays," *IEEE Trans. VLSI Syst.*, vol. 11, no. 1, pp. 44–54, 2003.

[3] M. Lin, A. E. Gamal, Y.-C. Lu, and S. Wong, "Performance benefits of monolithically stacked 3-d fpga," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 216–229, 2007.

[4] J. A. Valero, J. Septién, D. Mozos, and H. Mecha, "3d fpga resource management and fragmentation metric for hardware multitasking," in *IPDPS*, 2009, pp. 1–7.

[5] T. Marconi, Y. Lu, K. Bertels, and G. Gaydadjiev, "Online hardware task scheduling and placement algorithm on partially reconfigurable devices," in *ARC*, 2008, pp. 302–307.

[6] Y. Lu, T. Marconi, K. Bertels, and G. Gaydadjiev, "Online task scheduling for the fpga-based partially reconfigurable systems," in *ARC*, 2009, pp. 216–230.

[7] T. Marconi, Y. Lu, K. Bertels, and G. Gaydadjiev, "3d compaction: A novel blocking-aware algorithm for online hardware task scheduling and placement on 2d partially reconfigurable devices," in *ARC*, 2010, pp. 194–206.

[8] S. Yang, "Logic synthesis and optimization benchmarks user guide version 3.0," in *International Workshop on Logic and Synthesis*, 1991.

[9] K. Siozios and D. Soudris, "A power-aware placement and routing algorithm targeting 3d fpgas," *J. Low Power Electronics*, vol. 4, no. 3, pp. 275–289, 2008.