3D Compaction: A Novel Blocking-Aware Algorithm for Online Hardware Task Scheduling and Placement on 2D Partially Reconfigurable Devices

Thomas Marconi, Yi Lu, Koen Bertels, and Georgi Gaydadjiev

Computer Engineering Laboratory, EEMCS
TU Delft, The Netherlands
{thomas,yilu,koen,georgi}@ce.et.tudelft.nl
http://ce.et.tudelft.nl

Abstract. Few of the benefits of exploiting partially reconfigurable devices are power consumption reduction, cost reduction, and customized performance improvement. To obtain these benefits, one main problem needs to be solved is the task scheduling and placement. Existing algorithms tend to allocate tasks at positions where can block future tasks to be scheduled earlier denoted as "blocking-effect". To tackle this effect, a novel 3D total contiguous surface (3DTCS) heuristic is proposed for equipping our scheduling and placement algorithm with blocking-awareness. The proposed algorithm is evaluated with both synthetic and real workloads (e.g. MDTC, matrix multiplication, hamming code, sorting, FIR, ADPCM, etc). The proposed algorithm not only has better scheduling and placement quality but also has shorter algorithm execution time compared to existing algorithms.

1 Introduction

Hardware task scheduling and placement algorithms can be divided into two main classes: offline and online. Offline assumes that all tasks properties (e.g. task sizes, execution times, reconfiguration times, etc) are known in advance. The offline version can then do various optimizations before runtime. As a result, the offline version has a better performance than the online version. However, the offline version is not applicable for general multipurpose systems in which the properties of arriving tasks are unknown beforehand. In general multipurpose systems, the online version is needed.

In the offline version, the algorithm can make offline decisions. Hence, the time needed for making decisions in the offline version is not taken into account for the overall application time. In contrast, the online version needs to take decisions at runtime; as a result, the algorithm execution time is computed as an additional time for the overall application time. Therefore, the goal of the online version is not only to get better scheduling and placement quality but also to have a low runtime overhead.

Online scheduling and placement algorithms have to find a block of hardware resources for running each arriving task on a 2D partially reconfigurable device. 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. Here, the algorithms need to find the earliest starting time and free space for executing the task on the device in the future.

Many algorithms have been proposed to solve the scheduling and placement issue mentioned above, such as: Horizon [1], Stuffing [1], Classified Stuffing [2], Intelligent Stuffing [3], Reuse and Partial Reuse [4], Window-based Stuffing [5], and Compact Reservation [6]. However, none of them has a blocking-aware ability; the existing algorithms have a tendency to block future tasks to be scheduled earlier, referred as "blocking-effect". As a result, wasted area (volume), schedule time, and waiting time will increase significantly. To solve this problem, we propose a novel 3D total contiguous surface (3DTCS) heuristic to equip our algorithm with blocking-awareness. The goal of the proposed algorithm is not only to achieve better quality but also to have lower runtime overhead.

The main contributions of this paper are:

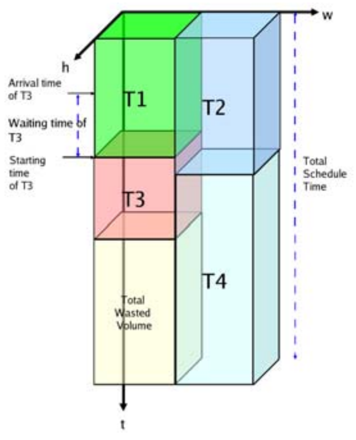
– the first blocking-aware online hardware task scheduling and placement algorithm;
– a novel 3D total contiguous surface (3DTCS) heuristic;
– a novel 3D Compaction (3DC) algorithm.

The rest of the paper is organized as follows. In Section 2, we introduce the problem of scheduling and placement on 2D area models. We give a short review of existing algorithms in Section 3. In Section 4, we present basic idea of our blocking-aware algorithm. The 3DTCS heuristic is described in Section 5. In Section 6, we present our proposed algorithm in detail. The algorithm is evaluated in Section 7. Finally, we conclude in Section 8.

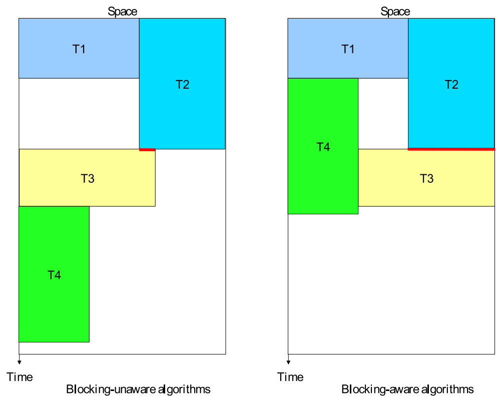
2 Problem of Scheduling and Placement on 2D Area Models

This problem definition is an extension of the definition of the problem of scheduling and placement on 1D area models as presented in [3]. Given a task set representing a multitasking application with their arrival times a_i, life-times lt_i, widths w_i and heights h_i, online scheduling and placement algorithms targeting the 2D area models of partially reconfigurable devices have to determine placements and starting times for the task set such that there are no overlaps in space and time among all tasks. The goals of the algorithms are: a) to utilize effectively the available FPGA resources (minimize wasted volume); b) to accelerate the overall application on the FPGA (minimize schedule time); c) to start executing arriving tasks on the FPGA earlier (minimize waiting time) and d) to keep the runtime overhead low (minimize the algorithm execution time).

We define the total wasted volume as the overall number of area-time units that are not utilized as illustrated in Figure 1(a). Total schedule time is the total



(a)



(b)

Fig. 1. Problem of scheduling and placement on 2D area models (a) and basic idea of blocking-aware algorithm (b)

number of time units for the execution of all tasks. Waiting time is the difference between starting and arrival times for each task (in time units). The algorithm execution time is the time needed to schedule and place the arriving task.

3 Related Work

In [1], Steiger et al. proposed the Horizon and Stuffing algorithms both for 1D and 2D area models. The Horizon guarantees that arriving tasks are only scheduled when they do not overlap in time or space with other scheduled tasks. The Stuffing schedules arriving tasks to arbitrary free areas that will exist in the future by imitating future task terminations and starts. In [1], the authors presented that the Stuffing outperforms the Horizon in scheduling and placement quality.

To tackle the drawback of the 1D Stuffing, Chen and Hsiung in [2] proposed their 1D Classified Stuffing. By classifying incoming tasks before scheduling and placement, the 1D Classified Stuffing performs better than the original 1D Stuffing.

In [3], Marconi et al. proposed their 1D Intelligent Stuffing to solve the problems of both the 1D Stuffing and Classified Stuffing. The main difference between their algorithm and previous 1D algorithms is the additional alignment flag of each free segment. The flag determines the placement location of the task within the corresponding free segment. By utilizing this flag, the 1D Intelligent Stuffing outperforms the previously mentioned 1D algorithms.

In [4], Lu et al. introduced their 1D reuse and partial reuse (RPR). The algorithm reuses already placed tasks to reduce reconfiguration time. As a result, the RPR outperforms the 1D Stuffing.

In [5], Zhou et al. proposed their 2D Window-based Stuffing to tackle the drawback of 2D Stuffing. By using time windows instead of the time events, the 2D Window-based Stuffing outperforms previous 2D Stuffing. The drawback of their 2D Window-based Stuffing is a high running time cost. To reduce the high runtime cost of Window-based Stuffing, they proposed their Compact Reservation (CR) in [6]. The main idea of the CR is the computation of the earliest available time (EA) matrix for every incoming task. That contains the earliest starting times for scheduling and placing the arriving task. The CR outperforms the original 2D Stuffing and their previous 2D Window-based Stuffing.

4 Basic Idea of Blocking-Aware Algorithm

Blocking-unaware algorithms do not care of their task placement positions whether they will block other incoming tasks or not in the future. They behave like drivers who park their vehicles wherever they want. Their parking places may be stumbling blocks for other drivers to park their cars. Figure 1(b) (left) illustrates the behavior of online scheduling and placement algorithms that do not have blocking-awareness. In this simple example, task T3 is becoming an obstacle for task T4 to be scheduled earlier.

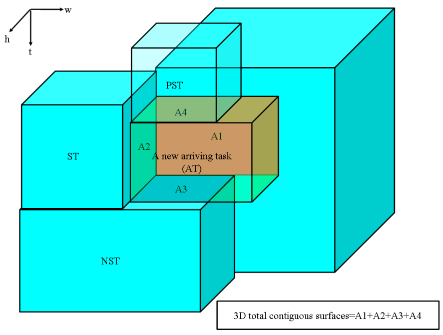
To tackle this problem, we introduce an algorithm that has an awareness to avoid placement that will be an obstacle for other future tasks. By placing task T3 to a different location as shown in Figure 1(b) (right), the proposed algorithm can avoid task T3 to be an encumbrance for task T4 to be started earlier. By scheduling T4 earlier, the FPGA can finish executing task T4 faster. To give the algorithm the necessary knowledge to avoid this "blocking-effect", the algorithm places tasks at locations as much as possible touching its prior tasks illustrated as bold lines on the figure. In next section, we will give a more detail explanation of this heuristic, termed 3D total contiguous surface (3DTCS).

5 3D Total Contiguous Surface (3DTCS) Heuristic

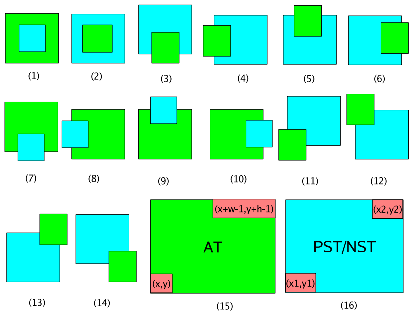
A hardware task on a 2D partially reconfigurable device using 2D area models can be illustrated as a 3D box. The first two dimensions are the required area (wh) on the device for running the task. The other dimension is the time dimension (t). To pack hardware tasks compactly during run time at the earliest time, we propose a new heuristic, named 3D total contiguous surface (3DTCS) heuristic.

The 3DTCS is the sum of all surfaces of an arriving task that is contacted with the surfaces of other scheduled tasks as depicted in Figure 2(a). The 3DTCS contains two components:

- the horizontal contiguous surfaces with previous scheduled tasks and next scheduled tasks;
- the vertical contiguous surfaces with scheduled tasks and the FPGA boundary.



(a)



(b)

Fig. 2. 3D total contiguous surface (3DTCS) heuristic (a) and horizontal contiguous surfaces (b)

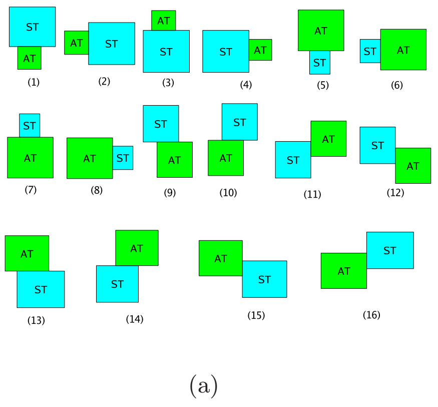
Table 1. Computations of horizontal contiguous surfaces for positions in Figure 2(b)(1)-(14)

Positions	Horizontal contiguous surfaces
(1)	$(x_2 - x_1 + 1)(y_2 - y_1 + 1)$
(2)	wh
(3)	$w(y + h - y_1)$
(4)	$(x + w - x_1)h$
(5)	$w(y_2 - y + 1)$
(6)	$(x_2 - x + 1)h$
(7)	$(x_2 - x_1 + 1)(y_2 - y + 1)$
(8)	$(x_2 - x + 1)(y_2 - y_1 + 1)$
(9)	$(x_2 - x_1 + 1)(y + h - y_1)$
(10)	$(x + w - x_1)(y_2 - y_1 + 1)$
(11)	$(x + w - x_1)(y + h - y_1)$
(12)	$(x + w - x_1)(y_2 - y + 1)$
(13)	$(x_2 - x + 1)(y_2 - y + 1)$
(14)	$(x_2 - x + 1)(y + h - y_1)$

In a simple example depicted in Figure 2(a), the horizontal contiguous surfaces with a previous scheduled task (PST) A4 and with a next scheduled task (NST) A3 in the figure give this heuristic an awareness on avoiding "blocking-effect"; while the other surfaces A1 and A2 (vertical contiguous surfaces) give this heuristic to better pack tasks in time and space. As a result, the proposed algorithm has a full 3D-view of the positions of all scheduled and placed tasks.

Intuitively, a higher 3DTCS value will result in more compaction both in space and time. This 3DTCS heuristic gives our proposed 3D compaction algorithm with blocking-aware ability to pack tasks better as it has a more complete view of all dimensions.

Figure 2(b)(1)-(14) and Table 1 show all the placement positions and their corresponding computations of horizontal contiguous surfaces. The arriving task (AT), with width w and height h, has a bottom-left coordinate (x, y) as shown in



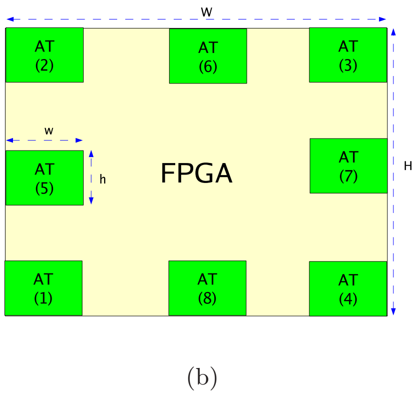


Fig. 3. Vertical contiguous surfaces with scheduled tasks (a) and the FPGA boundary (b)

Table 2. Computations of vertical contiguous surfaces with scheduled tasks for positions in Figure 3(a)(1)-(16)

Positions	Vertical contiguous surfaces with scheduled tasks
(1),(3)	$w.min(lt,(t_f - t_s))$
(2),(4)	$h.min(lt,(t_f - t_s))$
(5),(7)	$(x_2 - x_1 + 1).min(lt,(t_f - t_s))$
(6),(8)	$(y_2 - y_1 + 1).min(lt,(t_f - t_s))$
(9),(14)	$(x_2 - x + 1).min(lt,(t_f - t_s))$
(10),(13)	$(x + w - x_1).min(lt,(t_f - t_s))$
(11),(15)	$(y_2 - y + 1).min(lt,(t_f - t_s))$
(12),(16)	$(y + h - y_1).min(lt,(t_f - t_s))$

Table 3. Computations of vertical contiguous surfaces with the FPGA boundary for positions in Figure 3(b)(1)-(8)

Positions	Vertical contiguous surfaces with the FPGA boundary
(1)-(4)	$(w + h)lt$
(5),(7)	$h.lt$
(6),(8)	$w.lt$

Figure 2(b)(15). The arriving task can be contacted with the previous scheduled task (PST) and (or) the next scheduled task (NST) to produce the horizontal contiguous surfaces. The scheduled task has a bottom-left coordinate (x_1, y_1) and a top-right coordinate (x_2, y_2) as illustrated in Figure 2(b)(16).

The arriving task can be contacted with scheduled tasks and (or) FPGA boundary to produce the vertical contiguous surfaces. All placement positions of the arriving task (AT) and their corresponding computations of vertical contiguous surfaces with the scheduled task (ST) are shown in Figure 3(a) and Table 2. The arriving task with a life-time lt is started execution at time t_s; the finishing time of scheduled task is denoted as t_f. Computations of vertical contiguous surfaces between the arriving task with the FPGA boundary are illustrated in Figure 3(b) and formulated in Table 3.

6 The 3D Compaction (3DC) Algorithm

Figure 4 shows the pseudocode for the proposed 3D Compaction (3DC). The algorithm maintains two linked lists: the execution list and the reservation list. The execution list saves the information of all currently running tasks sorted in order of increasing finishing times; the reservation list contains the information of all scheduled tasks sorted in order of increasing starting times. The information stored in the lists are the bottom-left coordinate (x_1, y_1), the top-right coordinate (x_2, y_2), the starting time t_s, the finishing time t_f, the task name, the next pointer, and the previous pointer.

In lines 1-13, the algorithm computes the starting time matrix (STM) with respect to the arriving task area wh on the device area WH. The algorithm collects all possible positions that have enough space for the arriving task by scanning the executing and reservation lists. The algorithm fills each element

```
1. for (y=1;y<=H-h+1;y++)
{
    2. for (x=1;x<=W-w+1;x++)
    {
        3. STM(x,y)=a
    }
}

4. for all tasks in execution list
{
    5. for (y=max(1,$y_1$-h+1);y<=min($y_2$,H-h+1);y++)
    {
        6. for (x=max(1,$x_1$-w+1);x<=min($x_2$,W-w+1);x++)
        {
            7. if (STM(x,y) < $t_f$)
            {
                8. STM(x,y)=$t_f$
            }
        }
    }
}

9. for all tasks in reservation list
{
    10. for (y=max(1,$y_1$-h+1);y<=min($y_2$,H-h+1);y++)
    {
        11. for (x=max(1,$x_1$-w+1);x<=min($x_2$,W-w+1);x++)
        {
            12. if ((STM(x,y) < $t_f$) AND (STM(x,y)+lt>$t_s$))
            {
                13. STM(x,y)=$t_f$
            }
        }
    }
}
```

```
14. collect all positions from STM that have the earliest starting time
15. for all above positions
{
    16. c_3DTCS=compute 3D contact surfaces
    17. c_SFTD=compute sum of finishing time difference
    18. if (c_3DTCS>3DTCS_max AND c_SFTD<SFTD_min)
    {
        19. best_position=current position
        20. 3DTCS_max=c_3DTCS
        21. SFTD_min=c_SFTD
    }
    22. else if (c_3DTCS>3DTCS_max)
    {
        23. best_position=current position
        24. 3DTCS_max=c_3DTCS
    }
    25. else if (c_3DTCS=3DTCS_max AND c_SFTD<SFTD_min)
    {
        26. best_position=current position
        27. SFTD_min=c_SFTD
    }
}
28. if best_starting_time=arrival time
{
    29. add task to the execution list
}
else
{
    30. add task to the reservation list
}
31. if  the reservation list is not empty
{
    32. update reservation list
}
33. if  the execution list is not empty
{
    34. update execution list
}
```

Fig. 4. Pseudocode of 3D Compaction algorithm

of the STM with the arrival time of incoming task a (lines 1-3). The algorithm updates groups of elements that are affected by all executing tasks in execution list (lines 4-8) and by all scheduled tasks in reservation list (lines 9-13).

In line 14, 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 not only wants to get the best position in terms of starting time (time domain) but the best position in terms of space (space domain) as well. To pack compactly tasks, we propose to use the 3DTCS heuristic as presented earlier in Section 5. The algorithm computes the 3DTCS (line 16) using formulas from Table 1 to Table 3 and chooses the best position from all the best starting time positions. Hence, the algorithm does not need to compute the 3DTCS for all positions; it only computes the 3DTCS for the best positions (candidates) (line 15). Intuitively, the highest 3DTCS value gives the best position in terms of packing to avoid "blocking-effect".

Besides the 3DTCS heuristic, the algorithm also uses the sum of finishing time difference (SFTD) heuristic for all scheduled tasks that vertically contacted with the arriving task (referred as a VC set). The algorithm computes current SFTD ($c\_SFTD = \sum_{\forall tasks \in VC} |t_s + lt - t_f|$) in line 17. The SFTD heuristic gives our algorithm an ability to group tasks with similar finishing times to get large free space during deallocations.

The algorithm chooses the position with the highest 3DTCS value and the lowest SFTD value for allocating the arriving task (lines 18-27). Allocating the arriving tasks at the highest 3DTCS compacts the tasks both in time and space; while grouping tasks with similar finishing times creates more possibility to produce larger free space during deallocations.

The algorithm allocates the incoming task when there is available space for the task at its arrival time; otherwise, the algorithm needs to schedule the task for future execution. If the arriving task can be allocated at its arrival time (line 28), it will be executed immediately and added in the execution list (line 29); otherwise, it is inserted in the reservation list (line 30).

When the tasks in the reservation list are executed, they are removed from the reservation list and added in the execution list. The finished tasks in the execution list are deleted after execution. These updating processes are executed when the lists are not empty (lines 31-34).

The time complexity analysis of our 3DC is presented in Table 4. In which W, H, N_{ET}, N_{RT} are the FPGA width, the FPGA height, the number of executing tasks in the execution list, the number of reserved tasks in the reservation list, respectively.

The main difference between our algorithm and existing algorithms is the presence of the 3D compaction ability. Because of this 3D compaction ability, our algorithm can avoid "blocking-effect". In contrast, the existing algorithms do not have the blocking-awareness. Some existing algorithms only have the 2D compaction ability; instead, our algorithm has the 3D compaction ability to

Table 4. Time complexity analysis of 3D Compaction algorithm

Lines	Time Complexity
1-3	$O\,(WH)$
4-8	$O\,(WHN_{ET})$
9-13	$O\,(WHN_{RT})$
14	$O\,(WH)$
15-27	$O\,(WH)$
28-30	$O\,(max\,(N_{ET}, N_{RT}))$
31-32	$O\,(N_{RT})$
33-34	$O\,(N_{ET})$
Total	$O\,(WHmax\,(N_{ET}, N_{RT}))$

compact tasks both in time and space domains. Besides, the algorithm also has an ability to group tasks with similar finishing times to achieve larger free space during deallocations. In the CR, every element of their EA matrix is checked to know if it falls into the coverage rectangles of execution and scheduling tasks for updating as shown in [6]. In contrast, our algorithm updates the STM matrix in groups of elements affected by all executing (lines 5-6) and scheduled tasks (lines 10-11); the algorithm does not need to check each element for updating. As a result, our algorithm computes starting times faster than the CR. Moreover, our 3DC does not need to compute boundary values for all reconfigurable units of its free space in the periphery. As a consequence, our algorithm has less runtime overhead compared to the CR as will be presented later.

7 Evaluation

7.1 Evaluation in Terms of Scheduling and Placement Quality

Evaluation with Synthetic Workloads. We have built a discrete-time simulation framework in C to evaluate the proposed algorithm. The framework was compiled and run under Linux operating system on a Pentium-IV 3.4 GHz PC. To better evaluate the algorithm with synthetic workloads, (1) we modeled realistic random hardware tasks to be executed on a realistic target device; (2) we evaluated the algorithm not only in terms of scheduling and placement quality but also in terms of runtime overhead.

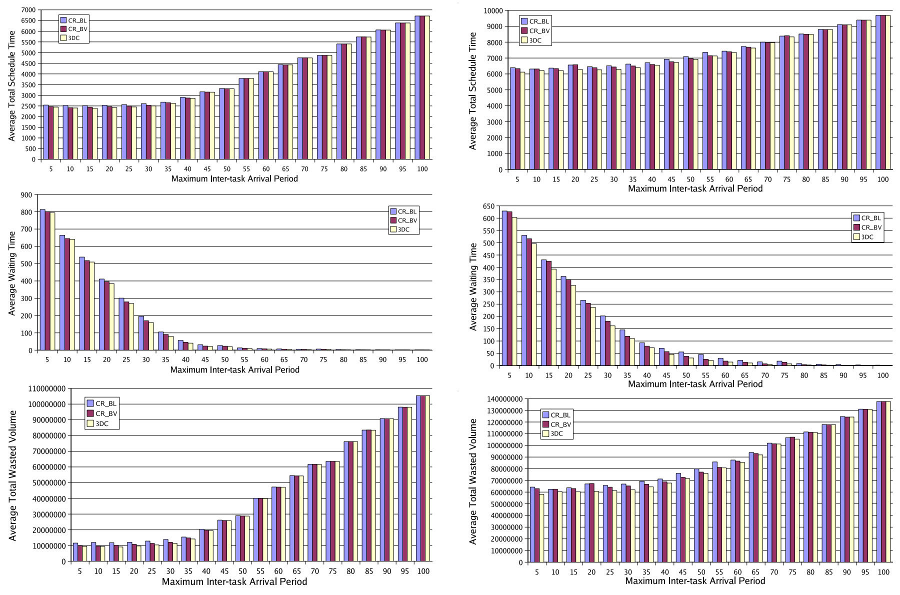
To model realistically the synthetic hardware tasks, we use a benchmark set (e.g. MDCT, matrix multiplication, hamming code, sorting, FIR, ADPCM, etc) from [10] and then use the DWARV [9] C-to-VHDL compiler to translate the benchmarks to VHDL. The VHDL code is then synthesized using the Xilinx ISE 8.2.01i_PR_5 tools to obtain the information of hardware task size range as a reference for our random task set generator. The task widths and heights are randomly generated 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. Every task set consists of 1000 tasks, each of which has a life-time and task size. The life-times are randomly generated in [5..100] time units, while the intertask-arrival periods are randomly chosen between one time unit and a specified maximum intertask-arrival period. Total tasks per arrival are randomly generated in [1..15]. Since the algorithms are online, the information of arriving tasks is unknown until their arrival times. We model a realistic FPGA with 116 columns and 192 rows of reconfigurable units (Virtex-4 XC4VLX200).

Our 3DC is designed for 2D area models. Therefore for fair comparison, we only compare our algorithm with algorithms that support 2D area models. Since the RPR [4], the Classified Stuffing [2], the Intelligent Stuffing [3] were designed only for 1D area models as shown in Section 3, we do not compare them with our 3DC.

Since the Stuffing outperforms the Horizon as presented in [1], we do not compare our algorithm to the Horizon. In [6], the CR outperforms the original 2D Stuffing [1] and the 2D Window-based Stuffing [5]; therefore, we only compare our algorithm to the CR.

To evaluate the 3DC, we have implemented three different algorithms: the CR [6] using BL (Bottom-Left) scheme (CR_BL), the CR [6] using BV (Boundary Value) scheme [7] (CR_BV), and our 3DC. The evaluation is based on three performance parameters as defined before in Section 2.

The CR does not have a blocking-awareness. Instead, our algorithm uses a 3D compaction for avoiding "blocking-effect". As a consequence, our algorithm has a better quality than the CR. The 3DC has up to 4.8 % less schedule time, 38.4 % less waiting time, and 22.9 % less wasted volume compared to the CR as shown in Figure 5(a).



(a) (b)

Fig. 5. Evaluation with synthetic (a) and real workloads (b)

The system idle time increases when the maximum inter-task arrival period increases; as a result, the average total schedule time and the average wasted volume increase.

The system is busier when the maximum inter-task arrival period decreases; tasks arrive more frequently to the system. Hence, it is more difficult to schedule tasks. Consequently, the average waiting time increases.

Evaluation with Real Workloads. To evaluate the 3DC with real workloads, realistic hardware tasks from [11] were used. 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 experimental results with real workloads are presented in Figure 5(b).

Figure 5(b) 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 up to 4.6 % less schedule time, 75.1 % less waiting time, and 9.9 % less wasted volume compared to the CR.

7.2 Evaluation in Terms of Algorithm Execution Time

To complete the evaluation, we also study the algorithm execution time since the execution time of online task scheduling and placement is considered as an overhead for the overall execution time of the applications. To show the effect of total number of scheduled and running tasks as well as FPGA area, we do simulation by changing these parameters as presented in Figure 6.

Figure 6 shows that our 3DC runs up to 133 times faster than the CR. The speed up will be higher for more scheduled and running tasks as well as for larger FPGA fabrics. Since the CR uses the boundary value heuristic for searching placement, the CR needs to compute boundary values for all reconfigurable units of its free space in the periphery. In contrast, our 3DC computes the 3DTCS only in one step as presented in Section 5. Moreover, the updating is done per each element of the matrix in the CR; each element is needed to be checked with all executing tasks and scheduled tasks. In contrast, our algorithm updates the matrix in groups of elements located by all executing tasks and scheduled tasks; the algorithm does not need to check each element for updating. As a result, our

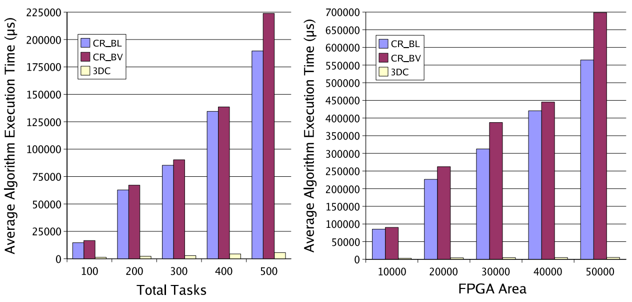


Fig. 6. Evaluation algorithm execution time

3DC has less runtime overhead than the CR by avoiding the CR's long boundary value computation and speeding up the starting times computation.

More FPGA area creates additional area suitable for the arriving task (more free volume) and more total number of scheduled and running tasks forces algorithms to check more tasks; as a result, the algorithms need more time to compute the matrix for finding starting time (all algorithms), all boundary values for all more candidates (CR algorithm) and all 3DTCS for all more candidates (3DC algorithm). Because of its long boundary value and matrix computations, the CR execution time increases faster than our 3DC.

8 Conclusions

To avoid "blocking-effect" of existing algorithms, we have proposed a new 3DTCS heuristic and used it to build a novel blocking-aware algorithm, 3D Compaction (3DC). Because of its 3D compaction, the 3DC places and schedules tasks more compactly. Moreover, the 3DC is equipped with an ability to group similar finishing time tasks to form larger free area for better allocating future tasks. Since the previous algorithm uses the boundary value heuristic for searching suitable placement, it needs to compute the values for all reconfigurable units of its free space in the periphery. In contrast, our 3DC computes the 3DTCS only in one step. In addition, the updating is done per each element of the matrix for finding starting time in the previous algorithm; each element is checked with all executing and scheduled tasks. Our 3DC updates the matrix in groups of elements located by all executing and scheduled tasks. The experimental results show that the 3DC not only has better scheduling and placement quality but also has lower runtime overhead compared to existing algorithms.

A possible direction for future research is to equip the algorithm with an ability to run tasks at different clock speeds or voltages for power saving (power-aware) and to place tasks based on required I/O positions (I/O-aware).

Acknowledgment

This work is sponsored by the hArtes project (IST-035143) supported by the Sixth Framework Programme of the European Community under the thematic area "Embedded Systems".

References

1. Steiger, C., Walder, H., Platzner, M.: Heuristics for Online Scheduling Real-Time Tasks to Partially Reconfigurable Devices. In: Y. K. Cheung, P., Constantinides, G.A. (eds.) FPL 2003. LNCS, vol. 2778, pp. 575–584. Springer, Heidelberg (2003)
2. Chen, Y., Hsiung, P.: Hardware Task Scheduling and Placement in Operating Systems for Dynamically Reconfigurable SoC. In: EUC, pp. 489–498 (2005)
3. Marconi, T., Lu, Y., Bertels, K.L.M., Gaydadjiev, G.N.: Online Hardware Task Scheduling and Placement Algorithm on Partially Reconfigurable Devices. In: Woods, R., Compton, K., Bouganis, C., Diniz, P.C. (eds.) ARC 2008. LNCS, vol. 4943, pp. 306–311. Springer, Heidelberg (2008)

4. Lu, Y., Marconi, T., Bertels, K.L.M., Gaydadjiev, G.N.: Online Task Scheduling for the FPGA-Based Partially Reconfigurable Systems. In: ARC, pp. 216–230 (2009)
5. Zhou, X., Wang, Y., Huang, X., Peng, C.: On-line Scheduling of Real-time Tasks for Reconfigurable Computing System. In: FPT, pp. 57–64 (2006)
6. Zhou, X., Wang, Y., Huang, X., Peng, C.: Fast On-line Task Placement and Scheduling on Reconfigurable Devices. In: FPL, pp. 132–138 (2007)
7. Sharma, D.D., Pradhan, D.K.: A Fast and Efficient Strategy for Submesh Allocation in Mesh-Connected Parallel Computers. In: IPDPS, pp. 682–689 (1993)
8. Xilinx: Virtex-4 FPGA Configuration User Guide. Xilinx user guide UG071 (2008)
9. Yankova, Y.D., Bertels, K.L.M., Vassiliadis, S., Meeuws, R.J., Virginia, A.: Automated hdl generation: Comparative evaluation. In: ISCAS, pp. 2750–2753 (2007)
10. Meeuws, R.J., Yankova, Y.D., Bertels, K.L.M., Gaydadjiev, G.N., Vassiliadis, S.: A Quantitative Prediction Model for Hardware/Software Partitioning. In: FPL, pp. 735–739 (2007)
11. Marconi, T., Lu, Y., Bertels, K.L.M., Gaydadjiev, G.N.: A Novel Fast Online Placement Algorithm on 2D Partially Reconfigurable Devices. In: FPT, pp. 296–299 (2009)