Online Hardware Task Scheduling and Placement Algorithm on Partially Reconfigurable Devices

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

Computer Engineering Laboratory, EEMCS
TU Delft, The Netherlands
{thomas,yilu}@ce.et.tudelft.nl, k.l.m.bertels@tudelft.nl,
g.n.gaydadjiev@ewi.tudelft.nl
http://ce.et.tudelft.nl

Abstract. In this paper, we propose an online hardware task scheduling and placement algorithm and evaluate it performance. Experimental results on large random task set show that our algorithm outperforms the existing algorithms in terms of reduced total wasted area up to 89.7%, has 1.5 % shorter schedule time and 31.3% faster response time.

1 Introduction

Reconfigurable devices with partial reconfigurable capabilities allow partial update of their hardware resources without interrupting the overall system operation [1]. Embedded applications which have exploited this capability include: neural network implementation [2], video communication [3], cryptography [4], crossbar switches [5], image processing [6], and more. Such functionality also allows multitasking applications on a single chip. However to fully exploit the advantages of such platforms the scheduling and placement problems are to be considered. This is to use the limited hardware resources as efficient as possible.

Our approach focusses on a number of shortcomings of existing algorithms in order to improve the FPGA resources utilization and improve the overall execution times. The main contributions of this paper are:

- a novel online scheduling and placement algorithm, called "Intelligent Stuffing";
- careful experimental evaluation of our and other existing algorithms based on statistically large 100k task sets randomly generated;
- improvements of up to 89.7% in terms of reduced total wasted area, 1.5% in schedule time and 31.3% shorter response time.

The remainder of this paper is organized as follows. The problems of scheduling and placement in dynamic reconfigurable devices is introduced in Section 2. In Section 3, we briefly discuss the previous art. Details of our algorithm are depicted in Section 4. In Section 5, we present the evaluation of the algorithm. Finally, we conclude the paper in Section 6.

R. Woods et al. (Eds.): ARC 2008, LNCS 4943, pp. 306–311, 2008.

2 Problem of Scheduling and Placement in Dynamic Reconfigurable Devices

Given a task set representing a multitasking application with their arrival times a_i, execution times e_i and widths w_i[1] , online scheduling and placement algorithms have to determine placements and starting times for the task set such as there are no overlaps both in space and time among all tasks. The goals of the algorithms are: a) to utilize effectively the available FPGA resources (referred as minimize wasted area in this paper); b) to run the overall application on FPGA faster (minimize schedule time); c) to shorten waiting time of the tasks to be executed on the FPGA (minimize response time) and d) to keep the runtime overhead low (minimize the algorithm execution time).

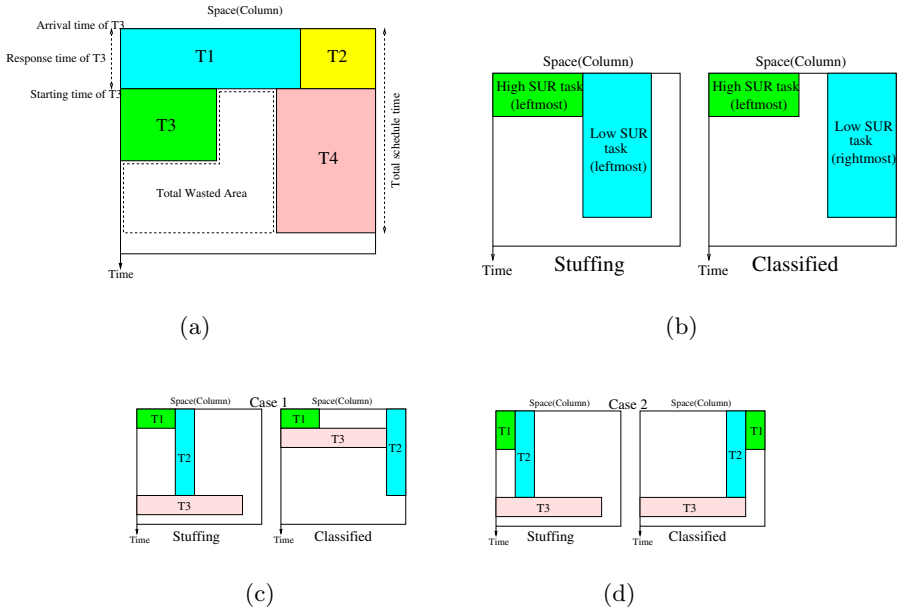


Fig. 1. Performance parameters and previous algorithms

We define total wasted area as the overall number of space-time units that are not utilized as shown in Figure 1(a). Total schedule time is the total number of time units for the execution of all tasks. Response time is the difference between starting and arrival times for each task (in time units). Total response time is the sum of response times for all tasks. The overall algorithm execution time is the cumulative time needed to schedule and place all the tasks.

[1] We use 1-D model for the FPGA as more representative for the current technology.

3 Previous Algorithms

In [7] [8], Steiger et al. proposed the Stuffing. It schedules tasks to arbitrary free areas that will exist in the future, including areas that will be used later by tasks currently in its reservation list. It always places a task on the leftmost of its free space as shown in the left of Figure 1(b). Because the Stuffing algorithm always places tasks on the leftmost edge of the available area, it places tasks T1 and T2 as shown in the left of Figure 1(c). These placements block task T3 to be scheduled earlier. In this case, it fails to place task T3 earlier.

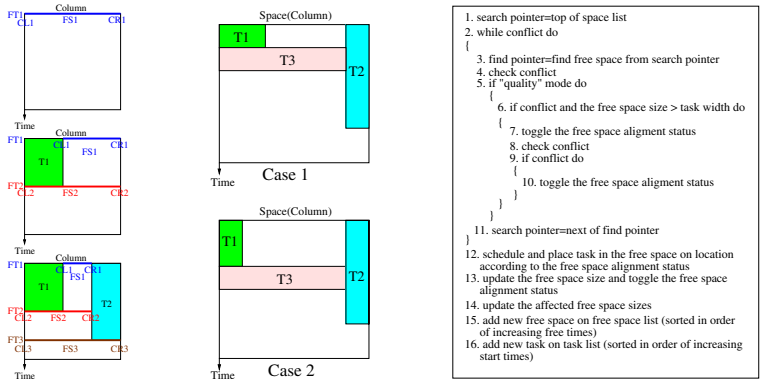
In [9], Chen and Hsiung proposed the Classified Stuffing to solve the drawback of the Stuffing in case 1 (Figure 1(c)). The main difference between the algorithm and the Stuffing is the classification of tasks. It can place a task on the leftmost or rightmost of its free space based on the task Space Utilization Rate (SUR). SUR is the ratio between the number of columns required by the task and its execution time. High SUR tasks (SUR $>$ 1) are placed starting from the leftmost available columns of the FPGA space, while low SUR tasks (SUR $\leq$ 1) are placed from the rightmost available columns as shown in the right of Figure 1(b). In case 1, it can recognize the difference between tasks T1 (high SUR task) and T2 (low SUR task), so it places successfully tasks on different placements. This makes task T3 earlier scheduling possible. However in case 2 (Figure 1(d)), it fails to solve the problem of the Stuffing. Because it doesn't recognize the difference between tasks T1 and T2 (both of the tasks are low SUR tasks), it fails to place tasks on different placements. These placements block task T3 to be scheduled earlier. Therefore in case 2, both of the previous algorithms fail to schedule task T3 earlier. Total wasted area, total schedule time, and total response time will increase as a consequence.

4 The Proposed Algorithm

Figure 2(a) (top) shows an empty FPGA and a leftmost alignment status is defined, e.g. a new free space always will be allocated at the leftmost position. At this point, the free space list SL contains only a single free space (FS_1) defined by its leftmost column (CL_1), its rightmost column (CR_1) and free time FT_1.

When a new task $T1$ arrives, the algorithm searches the free space list SL and places it on the leftmost edge of FS_1 (according to its alignment status). This action reduces the size of FS_1 as shown in the middle of Figure 2(a), toggles the alignment status of FS_1 from leftmost to rightmost, and creates a new free space FS_2. FS_2 has (CL_2, CR_2) dimension and its free time is FT_2 and leftmost alignment status.

Assume there is another task $T2$ simultaneously arriving with $T1$ the free space list SL will be processed again. Because the alignment status of FS_1 was changed to rightmost, $T2$ will be placed on rightmost edge of FS_1. This action reduces the FS_1 size as shown in Figure 2(a) (bottom) and again toggles the alignment status of FS_1 to leftmost. The size of FS_2 is also adjusted and a new free space FS_3 (CL_3,CR_3) is created with free time FT_3 and leftmost alignment



(a) (b) (c)

Fig. 2. Our algorithm

status. By keeping tasks $T1$ and $T2$ on the edges, the largest space possible is created, so future tasks can be scheduled earlier and we can address the problem of previous algorithms for both case 1 and case 2 as shown in Figure 2(b).

Our algorithm maintains two linked lists: a free space list (SL) and a task list (TL). The SL contains all free spaces FS_i with their previous pointers PP_i, dimensions (CL_i and CR_i), free times FT_i, alignment statuses AS_i and next pointers NP_i. The free time is the time when the corresponding free space can be used. The alignment status is a boolean determining the placement location of the task (leftmost or rightmost) within this free space segment. The new list entries of SL are inserted in order of increasing free times.

The TL stores all scheduled tasks with their previous pointers PP_j, start times ST_j, task dimensions (CL_j, CR_j), task execution times ET_j and next pointers NP_j. The start time is the time that the task initiates execution on the FPGA. The column left (CL_j) and right (CR_j) determine the FPGA area that is used by the task. The new list entries of TL are inserted in order of increasing of start times.

There are two operating modes: *speed* and *quality*. In the speed mode, the algorithm execution time is more important than the quality of scheduling and placement. While the quality mode is designed for higher utilization of the resources. The pseudocode of our algorithm is presented in Figure 2(c). When a new task arrives, our algorithm walks through the SL to find a first fit free space avoiding conflicts with the scheduled tasks in the TL (line 1 to 11). The first fit free space has the earliest free time which enough columns of reconfigurable units to fit the task.

If quality mode is chosen, lines 6 to 10 are executed for better quality (in speed mode those lines are skipped to reduce the algorithm execution time). In lines 6 to 8, a placement of the task at the opposite position to the alignment status is attempted. This action increases the quality, but it requires additional algorithm time. If the task still conflicts with the currently scheduled tasks in the TL (line 9), the alignment status of the corresponding free space is set to its initial condition (line 10).

In line 12, the first fit free space without conflicts with the TL list is found, however this space may be wider than that the task requirements. The task is placed on the FS_i edge according to its alignment status. As mentioned earlier, every placement changes the size and toggles the alignment status of the used free space (line 13). This action can also affect the other free space sizes (line 14) and adds a new free space in the SL (line 15) in addition to the new scheduled task in the TL (line 16).

The main difference between our algorithm and previously proposed algorithms is the additional alignment status of the free space segments and its handling. This status guides our algorithm to make the correct decision on task placement position in order to maximize the free space area and allow earlier placing of further tasks. In addition, our algorithm does not need to compute SUR, therefore it runs faster than the Classified Stuffing.

5 Evaluation

We implemented four different algorithms (the Stuffing [7] [8] (STF), the Classified Stuffing [9] (CTF) and our algorithm using *speed* mode (ISS) and *quality* mode (ISQ)) in ANSI-C and run them on a Pentium-IV 3.4 GHz PC using the same task sets. The simulated device consists of 96 columns to model Xilinx XCV1000 (96x64 reconfigurable units). The task widths and execution times are uniformly distributed in [1,96] columns of reconfigurable units and [1,1000] time units. We generate randomly 20 tasks for each task set and run all algorithms using 100,000 task sets. The evaluation is based on four performance parameters: total wasted area (TWA), total schedule time (TST), total response time (TRT), and total algorithm execution time (TAT) (μs).

Table 1. Experimentation results using 100,000 task sets

Performance parameters	STF	CTF	ISS	ISQ
TWA	1035449499	783069435	367934139	106709691
TST	651128773	648499814	644175488	641454400
TRT	335229077	317655028	276949454	230250447
TAT(μs)	2076694	2184614	2074848	2168651

Table 1 shows that even in *speed* mode our algorithm utilizes the FPGA better (decreasing the wasted area compared to the Stuffing by 64.5 %) with only 0.1 % algorithm time overhead used for saving the alignment status bit. In addition, it makes the overall application execution 1.1 % faster and has 17.4 % shorter waiting time. The *speed* mode is not only faster than the Classified Stuffing (5 % shorter algorithm execution time) but also utilizes the FPGA more effective by decreasing the wasted area by 53 %. Furthermore the application execution is reduced by 0.7 % with 12.8 % shorter total waiting time.

In *quality* mode the wasted area is decreased by 89.7 % compared to the Stuffing with only 4.2 % algorithm execution time overhead (saving the alignment status bit and finding alternative placements). Moreover it makes the application running 1.5 % faster with 31.3 % shorter total waiting time. In respect to the Classified Stuffing the *quality* mode is not only faster by 0.7 % in terms of algorithm

execution time but also decreases the FPGA wasted area by 86.4 %. Additionally, the overall application execution time is reduced by 1.1 % with 27.5 % better total waiting time.

6 Conclusions

In this paper we proposed a new algorithm for online task scheduling and show how it outperforms previous art in terms of reduced total wasted area, schedule time and response time. We also evaluated two different modes of our algorithm: the *quality* mode for better placement and scheduling quality and the *speed* mode the algorithm execution time is considered more important.

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. Lysaght, P., Dunlop, J.: Dynamic Reconfiguration of FPGAs. In: More FPGAs, pp. 82–94, EE&CS Books, Abingdon (1993)
2. Eldredge, J.G., Hutchings, B.L.: Density Enhancement of a Neural Network Using FPGAs and Run-Time Reconfiguration. In: Proceeding of IEEE workshop on FPGAs for custom computing machines, pp. 180–188 (1994)
3. Villasenor, J., Jones, C., Schoner, B.: Video Communications Using Rapidly Reconfigurable Hardware. IEEE Transactions on circuits and systems for video technology 5(6), 565–567 (1995)
4. Vuillemin, J., Bertin, P., Roncin, D., Shand, M., Touati, H., Boucard, P.: Programmable Active Memories: Reconfigurable Systems Come of Age. IEEE Transactions on VLSI Systems 4(1), 56–69 (1996)
5. Eggers, H., Lysaght, P., Dick, H., McGregor, G.: Fast Reconfigurable Crossbar Switching in FPGAs. In: Field-Programmable Logic: Smart Applications, New Paradigms and Compilers, pp. 297–306 (1996)
6. Wirthlin, M.J., Hutchings, B.L.: Sequencing Run-Time Reconfigured Hardware with Software. In: ACM/SIGDA International Symposium on Field Programmable Gate Arrays, pp. 122–128 (1996)
7. 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)
8. Steiger, C., Walder, H., Platzner, M.: Operating Systems for Reconfigurable Embedded Platforms: Online Scheduling of Real-Time Tasks. IEEE transaction on Computers 53(11), 1393–1407 (2004)
9. Chen, Y., Hsiung, P.: Hardware Task Scheduling and Placement in Operating Systems for Dynamically Reconfigurable SoC. In: Yang, L.T., Amamiya, M., Liu, Z., Guo, M., Rammig, F.J. (eds.) EUC 2005. LNCS, vol. 3824, pp. 489–498. Springer, Heidelberg (2005)