Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2014, Article ID 495080, 7 pages
http://dx.doi.org/10.1155/2014/495080

Research Article

An FPGA Task Placement Algorithm Using Reflected Binary Gray Space Filling Curve

Senoj Joseph Olakkenghil[1] **and K. Baskaran**[2]

[1] *Department of Electronics and Communication Engineering, Sri Krishna College of Technology, Coimbatore, Tamilnadu 641042, India*
[2] *Department of Computer Science Engineering, Government College of Technology, Coimbatore, Tamilnadu 641013, India*

Correspondence should be addressed to Senoj Joseph Olakkenghil; senoj_joseph@yahoo.com

Received 30 September 2013; Revised 30 December 2013; Accepted 24 January 2014; Published 16 April 2014

Academic Editor: Koen L. M. Bertels

With the arrival of partial reconfiguration technology, modern FPGAs support tasks that can be loaded in (removed from) the FPGA individually without interrupting other tasks already running on the same FPGA. Many online task placement algorithms designed for such partially reconfigurable systems have been proposed to provide efficient and fast task placement. A new approach for online placement of modules on reconfigurable devices, by managing the free space using a run-length based representation. This representation allows the algorithm to insert or delete tasks quickly and also to calculate the fragmentation easily. In the proposed FPGA model, the CLBs are numbered according to reflected binary gray space filling curve model. The search algorithm will quickly identify a placement for the incoming task based on first fit mode or a fragmentation aware best fit mode. Simulation experiments indicate that the proposed techniques result in a low ratio of task rejection and high FPGA utilization compared to existing techniques.

1. Introduction

Reconfigurable devices with partial reconfiguration capabilities allow multitasking applications on a single chip. Embedded applications like cryptography, video communication, image processing, and so forth can exploit this capability. Efficient placement and scheduling algorithm can improve FPGA resource utilization and overall execution time of applications.

One of the most interesting problems is to decide where to locate the bitmap of a new task in the FPGA when it must be run. A data structure is required to keep track of the available free area, and the algorithm must find out the best location for the arriving task, trying to use the reconfigurable area as efficiently as possible. In online placement system, due to dynamic addition and deletion of tasks, the empty area of FPGA becomes highly fragmented and FPGA area cannot be utilized efficiently.

In this paper, a new data structure based on one-dimensional run-length encoding is developed to manage the empty area. Using this data, structure placement algorithm can locate a suitable location to place the incoming task quickly. A new fragmentation metric gives an indication of continuity of free space. The FPGA surface is modeled by a matrix coded according to reflected binary gray curve. The results show significant improvement over placement using well-known algorithms like bottom left, 2D adjacency based placement, least interference fit technique, and CLook algorithm.

This paper is organized as follows. Section 2 presents an overview of the problem of scheduling and placement in dynamic reconfigurable devices. A brief review of various placement and scheduling techniques are given in Section 3. In Section 4, a new technique called reflected binary gray curve based placement is proposed. Section 5 describes the experimental setup made for performance analysis. Results of average device utilization, task rejection ratio, average task waiting time, and so forth are discussed in Section 6. Finally, conclusions are presented in Section 7.

2. Problem of Scheduling and Placement in Dynamic Reconfigurable Devices

The proposed online placement system model consists of host CPU and partially reconfigurable FPGA. The reconfigurable resources on FPGA are a set of CLB organized in a two-dimensional array. The placement module running on the host CPU consists of a scheduler, a placer, and a loader. The scheduler determines which of the tasks in the module library should be loaded and executed next. The placer will manage free space and find out the optimum placement for the task. The loader loads the configuration data of tasks in the FPGA. When a task is completed the resources occupied by it will be released.

The system assumes that the tasks arrive online. As long as free area is available in the FPGA, the incoming task will be placed in an unoccupied area on the FPGA. If there is no free space and the task cannot be delayed, then the task is rejected. A good placement algorithm should reduce rejection rate.

The tasks are nonpreemptive. Once a task is loaded into the FPGA, it runs to termination. The tasks should be independent without any precedence constraints. These task parameters are defined as follows: for a task $t_i = (h_i, w_i, a_i, s_i, d_i, x_i, y_i)$, h_i and w_i represent its height and width, respectively, and are measured in number of cells and a_i, s_i, and d_i are the task arrival time, execution time, and deadline time. The rectangular area is assigned to the task by its top left corner (x_i, y_i) where x_i is the row number and y_i is the column number. The size, arrival time, execution time, and deadline are uniformly distributed in a predefined region and a priori unknown.

3. Related Works

Bazargan et al. [1] proposed an algorithm for managing free space by keeping track of nonoverlapping rectangles. The main disadvantage is that the number of empty rectangles produced quickly increases with more task insertions. This can lead to some tasks being rejected even though there is adequate space to accommodate them but this space is divided between two nonoverlapping rectangles. To solve this problem, they presented the idea of allowing overlapping of the empty rectangles, specifically overlapping maximal empty rectangles MERs. For n tasks, we can have $O(n)$ nonoverlapping rectangles and, in the case of MERs, we can have $O(n^2)$ rectangles.

Walder et al. [2] proposed three partition algorithms based on Bazargan method: enhanced Bazargan, on the fly, and enhanced on the fly. The third is based on a 2D hashing table to find a feasible task placement with a run time complexity of $O(1)$, but they did not account for reconfiguration time and also they did not account for the update time needed to update the hashing table.

Ahmadinia et al. [3–6] proposed horizontal line algorithm in which two horizontal lines are used: one above and another below the placed tasks. They also presented a free space management based on the contour of the union of rectangles algorithm. Handa and Vemuri [7–9] proposed staircase algorithm for finding the maximal empty rectangles. The bottleneck is time for constructing staircase and finding MERs. Tabero et al. [10–12] used vertex lists to store free space where each vertex is a possible location for an input task. Tomono et al. [13] proposed a method in which module connectivity to the remainder of the system is taken into account. Jin et al. [14] proposed a set of algorithms called scan line algorithm. But finding maximum key elements and the MER is time consuming. Marconi et al. [15, 16] proposed an intelligent merging technique to speed up Bazargan algorithm without losing its placement quality. It is a combination of three techniques selected based on the task characteristics. The techniques are as follows: merge only if needed, partial merging, and direct combine. Deng et al. [17] proposed an algorithm which packs tasks densely called 2D and 3D adjacency method. Lee et al. [18, 19] proposed a CLook and CSAF method, also multistrategy fit algorithm. Bassiri and Shahhoseini [20] considered reconfiguration time by classifying tasks into significant or nonsignificant. Steiger et al. [21–23] proposed stuffing techniques for combined placement and scheduling. Belaid et al. [24] proposed an offline algorithm for placement of tasks. ELfarag et al. [25] and Esmaeildoust et al. [26] proposed various fragmentation aware techniques. Lu et al. [27, 28] proposed flow scan algorithm for placement of online tasks.

4. Proposed Work

The proposed work is based on a novel representation for vacant space inside FPGA. A data structure called run-length matrix has been introduced to describe the FPGA area. Run-length representation consists of a list of tuples. Each tuple (m, n) indicates an empty slot where m and n are the starting location and size of empty slot, respectively. In Figure 1, the area inside the dark shaded box indicates the task already placed. In this figure, three tasks T_1, T_2, and T_3 are already placed at locations 12, 54, and 24, respectively, on an FPGA of size 8×8. The remaining free area can be described using free space run-length matrix as shown below:

$$\text{FRL} = \{(0, 12), (16, 8), (32, 16), (56, 8)\}. \tag{1}$$

This representation is possible, because the FPGA cells are labeled using reflected binary gray space filling curve. This space filling curve has excellent spatial locality property. Therefore, when this array is mapped into one-dimensional array the run-length representation will be very compact. Secondly, the size of this depends only on the fragmentation level. The size of run-length list is independent of size of FPGA and the number of tasks running.

The width and height of the incoming task is assumed to be even. The algorithm first scans the run-length and identifies probable candidates for placement. For example, if the incoming task size is 4×4, it will first search the run-length matrix list for vacant space of 16 or more cells. The idea is that a 4×4 task placed at this location will occupy a single contiguous region. If it is not able to find such a location, then it will try to obtain a location which is a multiple of 8 and selected regions can be represented by two regions of 8

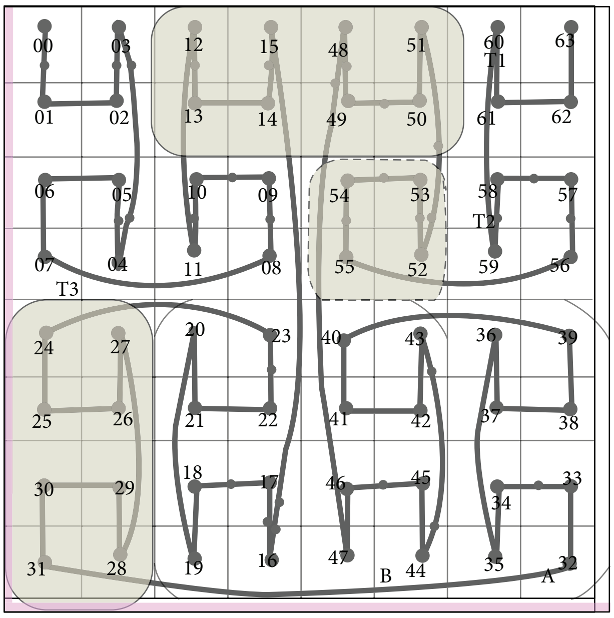

Figure 1: FPGA with some tasks already placed.

cells which are adjacent in 2D and so on. In order to avoid checking the same place again and again in the same instance the checked locations are stored in a list. For each probable location, the algorithm extracts a region of width and height equivalent to the incoming task (in this example 4×4). The region can be slided in the horizontal and vertical direction to get other possible locations. The extracted regions are analyzed to check whether they are vacant. In the above example, the algorithm finds two positions for placing the incoming task shown as A and B, in Figure 1. For placing at A, we need vacant space (32, 16) and placing at B requires vacant space TRL $= \{(16, 8), (40, 8)\}$. Based on resulting fragmentation one among these will be selected for placing the incoming task. If location A is selected, the FRL will be updated to $\{(0, 12), (16, 8), (56, 8)\}$.

Let r and c indicate the row and column of the candidate location cell X. Loop can be U shaped or inverted U shaped. Loop direction can be explored by checking the position of $X + 1$ and $X + 3$ using a look up reflected binary gray matrix. Each loop will have an entry which can be vertical or horizontal. This can be found by examining row and column of $X - 1$ cell. The U shaped loops at locations 12 and 48 have vertical entry of distance 4 and 8 rows, respectively. U shape loops at 24 and 40 have horizontal entry of length 4 column place. Similarly, we have inverted U loop with vertical entry at 16, 44, and so forth and horizontal entry at 32 and 56, respectively. This information will be useful while sliding task. When the task T_1 placed at position 12 get expired, here, again we find the blocks to be removed TRL $= \{(12, 4), (48, 4)\}$. The run-length matrix will be updated as follows FRL $= \{(0, 24), (56, 8)\}$.

In algorithms based on area matrix methods, whenever a new task is added or deleted the cells have to be recalculated. This takes a considerable amount of time. The run-length will be smaller in size (worst case will be one eighth of the number of CLB's) and hence less number of entries only need to be checked. Updating the run-length is also having less complexity.

The quality of placement algorithm can be improved by finding all feasible solutions and then selecting one based on fragmentation. Best fit finds the fragmentation index of all the feasible solutions and place the task in a position that reduces the resulting fragmentation. Due to the run-length representation, we make use of a new method to measure continuity of free space. Compared to other methods proposed in literature, this is faster and gives better results. Fragmentation is calculated using the method given by Gehr and Schneider [29]. Consider

$$F = 1 - \frac{\sum_{i=1}^{n} fi^{p}}{\left(\sum_{i=1}^{n} fi\right)^{p}} \quad p = 1, 2, \ldots, n. \tag{2}$$

Here, p is taken as 2. If the entire space is free, then fragmentation will be 0. In the worst case of checkerboard pattern, it will be almost 1.

The first fit method tries to place task in the first available location that can accommodate the incoming task. Best fit tries to fix the task in a place which reduces the overall fragmentation. It does not guarantee optimal results because it is a heuristic and the future inputs are unpredictable.

Mapping a task with odd dimension on to a reflected binary gray space will increase the fragmentation. To reduce complication, we consider the size of the task as the nearest even number. Therefore, the allotted space for the task will be slightly more than actual space required. This leads to internal fragmentation. In this paper, the tasks are assumed to have even tasks. The pseudocode is given below:

```
Input: incoming task $t_i$, Free space run-length FRL
Set Best_frag = 1, found = 0; Select $n$ such that $2^n \le w * h$ where $w$ and $h$ are width and height of the incoming task.
While $n > 0$ do
Check FRL for a vacant space of size more than $2^n$
Find a feasible location G inside the vacant space.
Select a region sufficient to occupy the incoming task and including G and represent it in run-length form TRL.
Try to insert TRL into FRL.
If any task already existing or the region exceeds FPGA boundary this will fail.
If fail then slide the region and try previous two steps.
If there is no overlap then insertion is success. If First fit then report G as the location for the incoming task and quit. If best fit algorithm update best_frag if the new fragmentation is low, set found = 1 and continue.
If success or fail that location will be stored in a list to avoid checking the same location again
Decrement $n$.
If found = 1
```

```
        Report the location of the incoming task
    else
        Return fail
    end
```

4.1. Complexity Analysis. Let m be number of empty slots in FRL and let g be the number of blocks to be inserted as in TRL. To find out the number of empty slots examined by the algorithm to place all the tasks, we consider the worst case which occur when the placed task splits the empty slot into two. Suppose all the blocks come inside the last slot. We examine $m - 1$ slots for fixing the blocks. While examining the mth slot, we place the first block creating new slot. We place the second block in this new slot creating another slot and so on. Therefore, by placing g blocks, we generate g new slots. Therefore, the total number of slots becomes $m + g$ of which the last slot created need not be examined to place task because all the m blocks have been placed. Hence, the loop needed to be run only up to a maximum of $m+g-1$ iterations. For best case loop needed to be run only g iterations. Complexity of finding fragmentation is $O(m)$. The clustering property assures that g and m will be small. Selecting regions and representing them into run-length are having complexity $O(1)$. Worst case complexity of sliding of the region is $O(w \times h)$ but w and h are width and height of incoming task and are small compared to size of FPGA.

To show that size of g is small, we calculated the size of TRL for blocks of all possible widths and heights on all possible locations. A histogram in Figure 2 is plotted for a 16×16 FPGA based on the size of TRL. From the figure, it is clear that in 90% of cases the size of TRL will be less than 5 and the average value is 3.905. This is true for bigger FPGA also. The maximum TRL size for a 8×8, 16×16, and 32×32 block on a 64×64 FPGA are 10, 22, and 46, respectively.

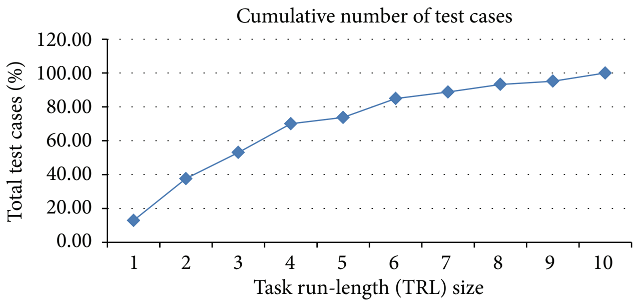

Figure 2: Cumulative graph of distribution of TRL size.

5. Experimental Setup

Simulation framework has been done using Matlab 7.8 running on 2.2 GHz Intel core i3 processor. The simulation is done using randomly generated data for evaluating the algorithm. This has been done in the past, because it is impossible to generate real data for future technological advancement. In this section, we present two methods: the first one is a fast placement (GFF) and the other is a fragmentation aware placement technique (GBF). These techniques are compared with standard placement techniques like bottom left, 2D adjacency based placement, least interference fit technique, and CLook algorithm. Bottom left (BL) is a classical bin packing algorithm which places the incoming task first empty slot available starting from bottom left corner of the FPGA. 2D adjacency based technique (Deng) chooses the location of the incoming tasks to make tasks placed "densely," in order to have a larger continuous free area remains. The 2D adjacency of a candidate cell is equal to the number of adjoining tasks/boundaries of the incoming task, if the base cell of the incoming task is placed here. The least interference technique (LIF) will select a location which minimizes the number of columns disturbed to minimize the number of running tasks getting halted during reconfiguration. CLook method is explained in Trong [14].

In order to evaluate the effectiveness of algorithm, simulation is performed for an FPGA with 16×16, 32×32, and 64×64 CLBs. The space filling curve requires the FPGA to be square shaped with dimension as a power of two. To demonstrate the impact of rejection rate on various parameters, we have used 16×16 FPGA. This model is adopted because the previous studies most relevant to this work used FPGA of similar size for their simulations and the space filling curve works on surface with size power of two. Sixty sets of 500 tasks each are randomly generated for each experimental environment and the results shown in the next section are the average over these sets. The height and width of the tasks are chosen randomly between 1 and a maximum value of 8 CLBs. The lifetime of the tasks is generated randomly between 1 and 500 time units. Delay between two consecutive tasks is also chosen between 1 and user defined L time units. The workload can be controlled using different upper bound L. A smaller L means that the tasks arrivals are more frequent, and FPGA area utilization is higher. All parameters are assigned by sampling a uniform random distribution function in their respective validity intervals. The proposed work uses a simple scheduling algorithm which can place task from a waiting list. The experiment was repeated for 32×32 and 64×64 size and the result seems to be similar and run-length size does not increase as FPGA is scaled.

The following assumptions are used in this work. The tasks are independent and preemptive. Preemptive tasks once started cannot be stopped before its expiry. Due to this, relocation of tasks is also not permitted. Since the tasks are independent, they can be scheduled in any order. Rotation of task is not used.

The following parameters are measured to test the effectiveness of the proposed algorithm. Suppose during the simulation interval $[0, T]$, N tasks arrived and n tasks were rejected. For a reconfigurable area of size $W * H$, consider the following:

(1) average task rejection ratio: a task may be rejected placement, if sufficient contiguous area is not available currently and it cannot meet its deadline, if scheduled at a later time:

$$\text{Average task rejection ratio} = \frac{n}{N} * 100\%; \qquad (3)$$

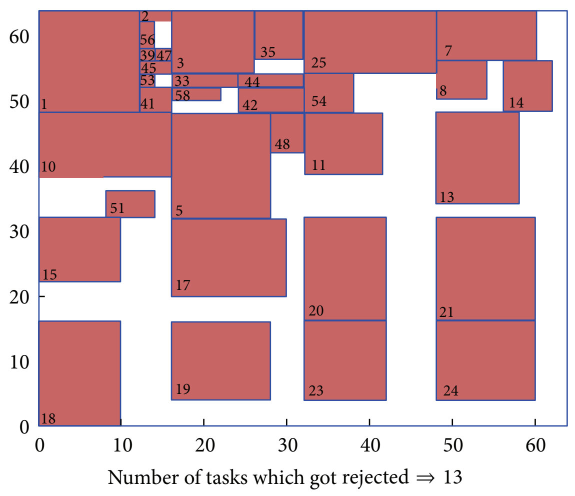

Figure 3: FPGA snapshot while placing tasks.

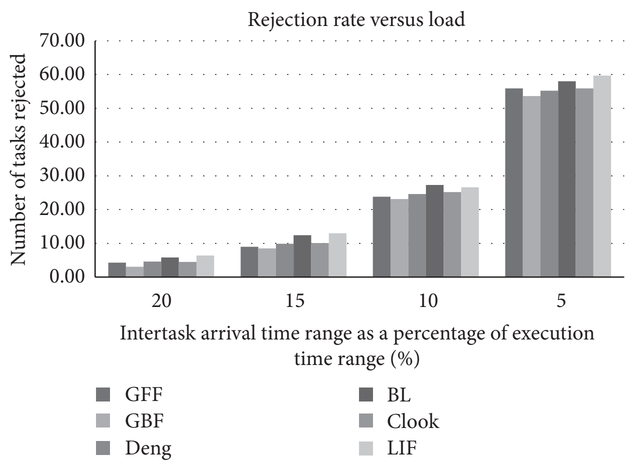

Figure 5: Rejection ratio for different values of load.

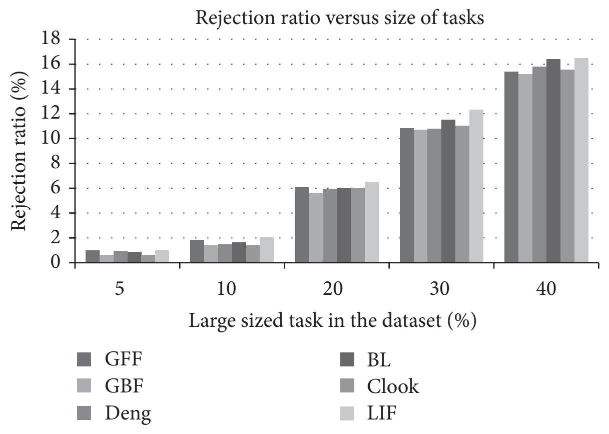

Figure 4: Variations of rejection rate with respect to percentage of large sized task in the dataset.

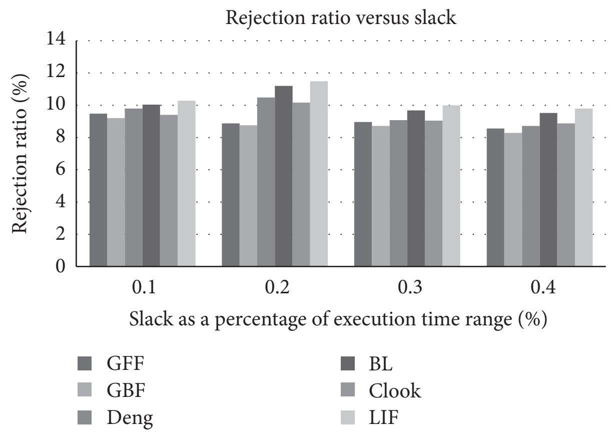

Figure 6: Rejection ratio decreases with increase in slack.

(2) total waiting time for tasks: if the online placement cannot find a feasible space the task will be added to a waiting list; when some task that is currently running is completed, the new space will be created and the waiting list will be examined to place tasks that can meet the deadline:

$$\text{Average area utilization} = \frac{\sum_i^{N-n} E_i * W_i * H_i}{T * W * H}. \quad (4)$$

Penalty ratio is the ratio of volume of rejected task to the total volume of all tasks. When a task gets rejected, the total free area in reconfigurable device is called wasted area. Good placement algorithm will have more utilization, less penalty ratio, less waste area, and less rejection ratio.

6. Results and Discussion

In this section, snapshot of simulation results of output at particular instance is shown in Figure 3. The coloured boxes correspond to tasks that are currently running.

A task that has been completed is not shown. The white region indicates empty region which is already getting fragmented due to placement and removal of tasks. The experiment is also repeated with skewed probability distribution of task's width and height to study the impact of task size on placement quality. Our placement method matches result with conventional methods. The rejection rate was more for the larger sized task as shown in Figure 4.

In the next experiment the intertask arrival time varied from 5% to 20% of execution time range. The rejection rate also increases with decrease in intertask arrival time range. When tasks arrive in quick succession, then more numbers of tasks will be running on the FPGA leaving less room for the newly arriving tasks. This is illustrated in Figure 5.

In order to examine the impact of deadline on the performance, we repeated the experiments with different values of slack. The deadline is calculated as the sum of arrival time, execution time, and slack.

When the deadline is tight, then more tasks get rejected. If the deadline is loose, then tasks can wait till their ALAP time and get placed whenever a free slot is available. When slack becomes very large, then none of the tasks gets rejected.

Table 1: Utilization factor average waiting time and execution time.

	16 × 16				32 × 32			64 × 64		
	Time (sec)	Rej	Wait	Util (%)	Time	Rej	Util (%)	Time	Rej	Util (%)
GFF	1.54	1.8	934	36.4	1.48	0	9	1.487	0	2.28
GBF	1.54	1.8	934	36.4	1.48	0	9.15	1.493	0	2.28
BL	0.173	5.6	1597	36	0.156	0	9.15	0.150	0	2.28
Deng	0.547	5.4	963	36	0.59	0	9.15	0.83	0	2.28
Clook	2.2	4	819	36.24	14.63	0	9.15	108.7	0	2.28
LIF	0.40	5.6	1530	35.9	0.54	0	9.15	0.829	0	2.28

Table 2: Performance metric for RBG code in first fit mode.

GFF algorithm	AT seconds	ARej	Await	Autil (%)	APen (%)	Awaste	FRL size	TRL size
Load40	0.62	0	0	27.27	0	0	46.81	3.95
Load30	0.67	0.2	2.4	33.94	0.17	0	58.87	3.84
Load20	2.47	11	228	46.40	3.8	1614	78.86	3.93
Load15	8.33	46.6	887	53.7	16.33	1576	89.44	3.75
Load10	17.57	91.4	1457	57.7	32.7	1447	96.67	3.65
Load05	32.09	169	1452	59.35	59.37	1196	96.19	3.15

Again the proposed method matches with existing methods as shown in Figure 6. Other results show that the average utilization for the reflected binary grey method is marginally better with lesser execution time than others.

Table 1 gives the performance of various algorithms. The waiting time is zero for 32 × 32 and 64 × 64 FPGA hence is not shown in the table. Even though BL, Deng, and LIF seem to be faster, their speed reduces when the size of the FPGA is increased. CLook has more execution time but its rejection rate performance is better than others. The proposed methods have rejection rate performance equal to CLook algorithm with faster execution time. Another feature of the proposed technique is that the execution time increases less rapidly when the FPGA size is scaled up. For CLook, the time taken will be very slow for bigger FPGAs.

Table 2 lists average algorithm execution time, average number of tasks rejected, average waiting time for the tasks, average utilization ratio, average penalty ratio, average waste area, average size of FRL, and average size of TRL obtained by simulating a 64 × 64 FPGA. The test dataset load05 means that the intertask time interval is [1 to 5] time units. Results show that the utilization ratio increases with load but flattens beyond some particular value. Waste area decreases with increase in utilization ratio. Waiting time, algorithm execution time, and average wait time increases with increase in load. Another important finding is that the average size of FRL and task run-length (TRL) are very small even though their theoretical values are high.

7. Conclusions

In this paper, a new approach for scheduling and placement of task on a dynamic reconfigurable device based on reflected binary gray space filling curve method is being presented with the goal of minimizing task rejection ratio and increasing FPGA utilization. The free space is managed as one-dimensional run-length based representation. Also, a new method to find the fragmentation is used. The algorithm does not consider routability, I/O communication, and heterogeneous FPGA. The algorithm can be improved to reduce the total reconfiguration overhead by reusing some of the task locations. Hence tremendous opportunities exist for research in this area.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

The authors thank the SKCT management and acknowledge the immense help received from the scholars whose articles are cited and included in references of this paper. The authors are also grateful to authors/editors/publishers of all those articles, journals, and books from where the literature for this paper has been reviewed and discussed.

References

[1] K. Bazargan, R. Kastner, and M. Sarrafzadeh, "Fast template placement for reconfigurable computing systems," *IEEE Design and Test of Computers*, vol. 17, no. 1, pp. 68–83, 2000.

[2] H. Walder, C. Steiger, and M. Platzner, "Fast online task placement on FPGAs: free space partitioning and 2D-hashing," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS '03)*, p. 178, IEEE-CS Press, 2003.

[3] A. Ahmadinia and J. Teich, "Speeding up on-line placement for Xilinx FPGA by reducing configuration overhead," in *Proceedings of the International Conference on Very Large Scale*

Integration of System on Chip (VLSI-SoC '03), pp. 118–122, Bavaria, Germany, December 2003.

[4] A. Ahmadinia, C. Bobda, M. Bednara, and J. Teich, "A new approach for on-line placement on reconfigurable devices," in *Proceedings of the 18th International, Parallel and Distributed Processing Symposium (IPDPS '04)*, pp. 134–140, 2004.

[5] A. Ahmadinia, C. Bobda, and J. Teich, "A dynamic scheduling and placement algorithm for reconfigurable hardware," in *Proceedings of the International Conference on Architecture of Computing Systems (ARCS '04)*, pp. 125–139, 2004.

[6] A. Ahmadinia, C. Bobda, S. P. Fekete, J. Teich, and J. C. van der Veen, "Optimal free-space management and routing-conscious dynamic placement for reconfigurable devices," *IEEE Transactions on Computers*, vol. 56, no. 5, pp. 673–680, 2007.

[7] M. Handa and R. Vemuri, "An integrated online scheduling and placement methodology," in *Proceedings of the International Conference on Field Programmable Logic and Application*, pp. 444–453, Leuven, Belgium, August 2004.

[8] M. Handa and R. Vemuri, "Area fragmentation in reconfigurable operating systems," in *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms*, pp. 77–83, CSREA, June 2004.

[9] M. Handa and R. Vemuri, "An efficient algorithm for finding empty space for online FPGA placement," in *Proceedings of the 41st Design Automation Conference (DAC '04)*, pp. 960–965, June 2004.

[10] J. Tabero, J. Septién, H. Mecha, and D. Mozos, "Task placement heuristic based on 3D-adjacency and look-ahead in reconfigurable systems," in *Proceedings of the Asia and South Pacific Design Automation Conference*, pp. 396–401, January 2006.

[11] J. Tabero, J. Septién, H. Mecha, and D. Mozos, "A low fragmentation heuristics for task placement in 2D RTR hardware management," in *Proceedings of the 14th International Conference on Field Programmable Logic and Application (FPL '04)*, Lecture Notes in Computer Science, pp. 241–250, Leuven, Belgium, September 2004.

[12] J. Tabero, J. Septien, H. Mecha, and D. Mozos, "Vertex list approach to 2D HW multitasking management in RTR FPGAs," in *Proceedings of the Conference on Design of Circuits and Integrated Systems (DCIS '03)*, pp. 545–550, Ciudad Real, Spain, November 2003.

[13] M. Tomono, M. Nakanishi, S. Yamashita, K. Nakajima, and K. Watanabe, "A new approach to online FPGA placement," in *Proceedings of the 40th Annual Conference on Information Sciences and Systems (CISS '06)*, pp. 145–150, Princeton, NJ, USA, March 2006.

[14] C. Jin, D. Qingxu, H. Xiuqiang, and G. Zonghua, "An efficient algorithm for online management of 2D area of partially reconfigurable FPGAs," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE '07)*, pp. 1–6, April 2007.

[15] T. Marconi and T. Mitra, "A novel online hardware task scheduling and placement algorithm for 3D partially reconfigurable FPGAs," in *Proceedings of the International Conference on Field-Programmable Technology (FPT '11)*, pp. 1–6, New Delhi, India, December 2011.

[16] T. Marconi, Y. Lu, K. Bertels, and G. Gaydadjiev, "Intelligent merging online task placement algorithm for partial reconfigurable systems," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '08)*, pp. 1346–1351, March 2008.

[17] Q. Deng, F. Kong, N. Guan, L. Mingsong, and W. Yi, "Online placement of real time tasks on 2D partially run time reconfigurable FPGAs," in *Proceedings of the 5th IEEE International Symposium on Embedded Computing (SEC '08)*, pp. 20–25, 2008.

[18] T.-Y. Lee, C.-C. Hu, and C.-C. Tsai, "Adaptive free space management of online placement for reconfigurable systems," in *Proceedings of the International MultiConference of Engineers and Computer Scientists (IMECS '10)*, vol. 1, pp. 322–326, Hong Kong, March 2010.

[19] T. Y. Lee, C. C. Hu, and C. C. Tsai, "Multi-strategy online placement for dynamically partial reconfigurable device," in *Proceedings of the International Conference on High-Speed Circuits Design*, pp. H-20–H-26, October 2009.

[20] M. M. Bassiri and H. S. Shahhoseini, "A new approach in on-line task scheduling for reconfigurable computing systems," in *Proceedings of the 21st IEEE International Conference on Application-Specific Systems, Architectures and Processors*, pp. 321–324, July 2010.

[21] C. Steiger, H. Walder, and M. Platzner, "Operating systems for reconfigurable embedded platforms: online scheduling of real-time tasks," *IEEE Transactions on Computers*, vol. 53, no. 11, pp. 1393–1407, 2004.

[22] C. Steiger, H. Walder, M. Platzner, and L. Thiele, "Online scheduling and placement of real-time tasks to partially reconfigurable devices," in *Proceedings of the 24th IEEE International Real-Time Systems Symposium (RTSS '03)*, pp. 224–235, Cancun, Mexico, December 2003.

[23] C. Steiger, H. Walder, and M. Platzner, "Heuristics for online scheduling real-time tasks to partially reconfigurable devices," in *Proceedings of the 13th International Conference on Field Programmable Logic and Application (FPL '03)*, pp. 575–584, Lisbon, Portugal, September 2003.

[24] I. Belaid, F. Muller, and M. Benjemaa, "Off-line placement of hardware tasks on FPGA," in *Proceedings of the 19th International Conference on Field Programmable Logic and Applications (FPL '09)*, pp. 591–595, Prague, Czech Republic, September 2009.

[25] A. A. Elfarag, H. M. El-Boghdadi, and S. I. Shaheen, "Fragmentation aware placement in reconfigurable devices," in *Proceedings of the 6th IEEE International Workshop on System on Chip for Real Time Applications (IWSOC '06)*, pp. 37–44, December 2006.

[26] M. Esmaeildoust, M. Fazlali, A. Zakerolhosseini, and M. Karimi, "Fragmentation aware placement algorithm for a reconfigurable system," in *Proceedings of the 2nd International Conference on Electrical Engineering*, pp. 1–5, March 2008.

[27] Y. Lu, T. Marconi, G. Gaydadjiev, and K. Bertels, "An on-line task placement algorithm for partially reconfigurable systems," in *Proceedings of the Architecture and Compiler for Embedded Systems (ACES '07)*, Edegem, Belgium, September 2007.

[28] Y. Lu, T. Marconi, G. Gaydadjiev, and K. Bertels, "An efficient algorithm for free resources management on the FPGA," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '08)*, pp. 1095–1098, Munich, Germany, March 2008.

[29] J. Gehr and J. Schneider, "Measuring fragmentation of two-dimensional resources applied to advance reservation grid scheduling," in *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID '09)*, pp. 276–283, May 2009.