A Fast Algorithm for Finding Maximal Empty Rectangles for Dynamic FPGA Placement

Manish Handa and Ranga Vemuri
Department of ECECS, University of Cincinnati,
Cincinnati, OH 45221-0030, USA.
{mhanda,ranga}@ececs.uc.edu

Abstract

In this paper, we present a fast algorithm for finding empty area on the FPGA surface with some rectangular tasks placed on it. We use a staircase *datastructure to report the empty area in the form of a list of maximal empty rectangles. We model the FPGA surface using an innovative encoding scheme that improves runtime and reduces memory requirement of our algorithm. Worst-case time complexity of our algorithm is $O(xy)$ where x is number of columns, y is number of rows and x.y is the total number of cells on the FPGA.*

1. Introduction

In a partially reconfigurable systems, an application is divided into smaller tasks that run concurrently on the FPGA. After a task finishes execution, it is removed from the FPGA and the area can be reclaimed and reused by next set of tasks. Since placement takes place at run-time of the application (online placement), the placement time is an overhead on total execution time of the application. In such a scenario, a fast algorithm to maintain empty area on the FPGA can help in fast placement of subsequent tasks.

Bazargan et al. [1] proposed heuristics for maintaining empty area in the form of non-overlapping rectangles. Walder et al. [3] improved the quality of placement by delaying free space partitioning. In both cases, free space is partitioned into non-overlapping rectangles, which is not optimal. Such a partitioning degrades the quality of placement. Edmonds et al. proposed an algorithm for searching empty space in large two-dimensional data-sets [2].

2. Model of a Dynamically Reconfigurable FPGA

In our model, each task is viewed as a hard rectangular macro with fixed orientation and finite height and width. The FPGA surface is modeled as a two dimensional array, with X number of columns and Y number or rows, called *area matrix*. Each cell in the array represents a CLB in the FPGA. Figure 1 shows the *area matrix* of a FPGA with two tasks placed on it. Each cell in the *area matrix* has a weight

1	−5	−4	−3	−2	−1	1	1	1	1
2	−5	−4	−3	−2	−1	2	2	2	2
3	−5	−4	−3	−2	−1	3	3	3	3
4	−5	−4	−3	−2	−1	4	4	4	4
5	1	1	1	1	1	5	5	5	5
6	2	2	2	−4	−3	−2	−1	6	6
7	3	3	3	−4	−3	−2	−1	7	7
8	4	4	4	−4	−3	−2	−1	8	8
9	5	5	5	1	1	1	1	9	9

(0,0)

Figure 1. *Modeling FPGA area Matrix*

associated with it. Each empty cell is represented by a positive number and every occupied cell is represented by a negative number. A positive number in a cell gives number of contiguous empty cells above and including that cell in that column and a negative number in an occupied cell gives the remaining width of the task measured in the right direction from that cell.

3. Finding Maximal Empty Rectangles

Empty space on the surface of a FPGA, with some rectangular tasks placed on it, can be covered using a finite set of rectangles called empty rectangles. A Maximal Empty Rectangle (MER) is the empty rectangle that cannot be covered fully by any other empty rectangle. In this paper, we propose a fast and efficient algorithm for finding empty space in a partially reconfigurable FPGA. The empty space is maintained as a list of overlapping maximal rectangles. Our algorithm can be used for finding empty rectangles for fast placement, defragmentation and task relocation.

A *staircase*(x,y) is defined as the the collection of all overlapping empty rectangles with (x,y) as their lowest

1530-1591/04 $20.00 (c) 2004 IEEE

right virtex. Figure 2 shows an example of *staircases* on an *area matrix*. More details on *staircase* data structure can be

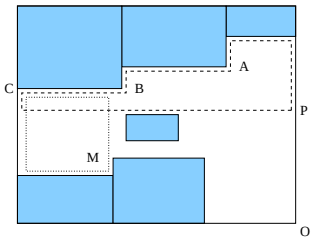

Figure 2. *Example of Staircases on a FPGA*

found in [2]. However, [2] uses a completely different encoding based on 0's and 1's. Our new encoding method described above leads to significant speedups as we will see later in this paper. Note that any maximal empty rectangle will be contained in one and only one staircase.

A *staircase* at point $(x+1,y)$ can be easily constructed from a *staircase* at point (x,y). Construction of *staircases* is described in [2] in detail.

We make *staircases* at empty locations in the *area matrix* and extract maximal empty rectangles out of them.

3.1. Extracting Maximal Empty Rectangles from a Staircase

In order to make staircases, we scan the *area matrix* row-by-row basis from top to bottom. Each row is scanned from left to right direction.

A *staircase* is possible at an empty location only. In [2], each location is checked to see if that is empty. In our application, the occupied cells exist in a cluster. We use this fact to improve runtime for our algorithm. If a negative number is encountered while scanning the *area matrix* from left to right in a row, some cells are skipped. This is because, at the left boundary of a task, the negative integer *i* gives width of the task. So, $|i|$ number of cells to the right are occupied by the same task and can be skipped. This gives considerable savings in run time.

In order to make a *staircase* at location $(x+1,y)$, we need to calculate the number of contiguous empty cells (y" - y) in column $x+1$. In [2], this number was calculated by adding one to the height of free cells in previous row at the same column (at location x+1,y+1). So, height information for a full row of data cells need to be kept in memory. In our approach, the positive weight of a cell gives the number of empty locations above that cell directly. As a result we do not need to save height information of the previous row. So, our algorithm needs half the memory as compared to [2].

A *staircase* is maximal if it cannot be extended down or to its right. If the vertical edge of the *staircase* can be extended to the right, the new rectangles in the new *staircase* will cover all the old rectangles. Same is true for the bottom edge of the *staircase*. We check only *maximal staircases* for extracting maximal empty rectangles.

Whether the bottom edge or the right edge of the *staircase*(x,y) can be extended depends upon location of occupied cells in column x+1 and row y-1. Let x^* be the X-coordinate of leftmost positive entry of a horizontal block of positive entries starting at $(x,y-1)$. Let Let y^* be the Y-coordinate of topmost positive entry of a vertical block of positive entries starting $(x+1,y)$. Consider a step in *staircase*(x,y) with top left corner (x_i,y_i). The rectangle (x_i,x,y_i,y) is maximal if and only if $x_i < x^*$ and $y_i > y^*$ [2].

After a *staircase* is constructed at a point, all the rectangles in the *staircase* can be checked to see if they are maximal empty rectangles. All maximal rectangles are reported and the algorithm proceeds to next point.

4. Time and Space Complexity

An FPGA has x number of columns and y number of rows. In the worst case, a staircase need to be made at every column in every row. A staircase can be made in constant time. A staircase can be checked for maximal rectangles in a very small amount of time, considered to be a constant [2]. So, worst case performance of our algorithm is $O(xy)$.

Space complexity of our algorithm is $O(min(x,y))$. But as discussed in Section 3.1, our algorithm uses only half as much memory compared to [2].

5. Acknowledgments

This work is sponsored in part by the Dayton Area Graduate Studies Institute (DAGSI) and the Air Force Research Laboratory (AFRL) research program under contract number IF-UC-00-07.

References

[1] K. Bazargan, R. Kastner, and M. Sarrafzadeh. Fast Template Placement for Reconfigurable Computing Systems . In *IEEE Design and Test - Special Issue on Reconfigurable Computing*, volume 17(1), pages 68–83, Jan.-Mar. 2000.

[2] J. Edmonds, J. Gryz, D. Liang, and R. J. Miller. Mining for Empty Spaces in Large Data Sets. *Theoretical Comput. Sci.*, 3(296):435–452, 2003.

[3] H. Walder, C. Steiger, and M. Platzner. Fast Online Task Placement on FPGAs: Free Space Partitioning and 2D-Hashing . In *International Parallel and Distributed Processing Symposium (IPDPS'03)* , page 178, April 2003.