Heterogeneous FPGA Floorplanning Based on Instance Augmentation

Rong Liu **Sheqin Dong** **Xianlong Hong**

Department of Computer Science & Technology, Tsinghua University, Beijing, China

Abstract

FPGA devices are now fabricated in advanced, ultra deep sub-micron technology with multi-million gate capacity and will be more complexity in the future. As a result, FPGA floorplanning will be extremely important in mapping designs to modern FPGAs. With the heterogeneous logic and routing resources on the device, floorplanning for modern FPGA is quite different from the well researched ASIC floorplanning. This paper presents an efficient algorithm addressed to heterogeneous FPGA floorplanning. The proposed algorithm is quite different from other simulated annealing based algorithms in that it employs an instance augmentation method to control the stochastic optimization. Experiments show that the proposed algorithm can generate floorplans for Xilinx's *XC3S5000* architecture successfully and very fast in most cases.

Keywords: floorplan, FPGA, instance augmentation.

1. Introduction

The advent of multi-million gates devices with heterogeneous logic resources makes it possible to implement much more applications with FPGA. On the other hand, these large design sizes significantly impact cycle time and make hierarchical approaches based on partition and floorplanning need to be introduced in the FPGA physical design flow [1].

As a key ingredient in hierarchical flows, FPGA floorplanning is becoming extremely important. Since there are heterogeneous logic and routing resources on modern FPGA devices and the number of each kind of resources is fixed, FPGA floorplanning can be formulated as a fixed-outline problem which shapes the modules and places them inside the outline of the target device to make their requirements for different kinds of resources satisfied so that the total wirelength minimized.

Many algorithms for ASIC floorplanning were proposed in the last few decades [3, 4, 5, 6, 7]. But all these algorithms treat the floorplanning resources as homogeneous, which means that no consideration of the modules' requirements for different kinds of resources is taken into. Hereby, these algorithms can not be used in heterogeneous FPGA floorplanning.

This work is partly supported by NSFC 60473126, NSFC and RGC of Hong Kong foundation 60218004, Hi-Tech Research & Development (863) Program of China 2004AA1Z1050.

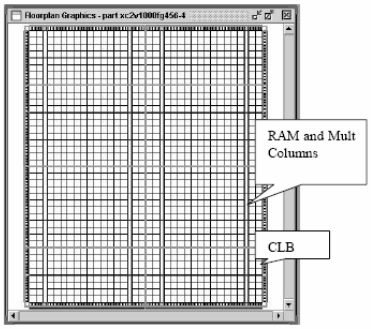

Figure 1. The heterogeneous FPGA architecture.

The first floorplanning algorithm targeted for heterogeneous FPGAs was presented in [2]. It uses slicing trees to represent floorplans and perturb slicing trees in the control of simulated annealing. Despite of its innovation, this algorithm is kind of slow.

In this paper, we present an efficient algorithm addressed to heterogeneous FPGA floorplanning. The proposed algorithm uses sequence pair to represent floorplans and applies an instance augmentation method. It starts with a small instance composed of a part of the given modules and then augments this instance step by step until all the modules are floorplanned. This method is somewhat like the branch-and-bound technique in that it only augments instance when the solution of current instance is promising, which means its partial floorplan is feasible. Experiments show that such an instance augmentation method can handle heavily constrained floorplanning problem.

The proposed algorithm focuses on the FPGA architecture described in [1], which has columns of CLBs with column pairs of RAMs and Multipliers interleaved. Fig.1 illustrates the architecture of this kind of heterogeneous architecture. Experiments based on Xilinx's *XC3S5000* (the largest chip of *Spartan3* family) show that in most cases, the proposed algorithm can generate floorplans feasible for the target device quite fast.

2. Problem Definition

A module in FPGA floorplanning is associated with a resource requirement vector $r=(n_c, n_r, n_m)$, indicating that

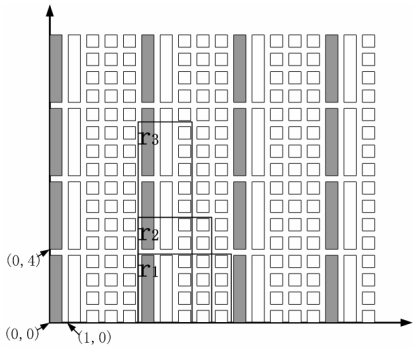

Figure 2. A simplified FPGA architecture

the module requires n_c CLBs, n_r RAMs, and n_m Multipliers. Given a set of modules and their connections, the objective of floorplanning is to place and shape each module inside the chip so as to fulfill its resource requirements, assure no modules overlapping with each other, and a given cost is optimized.

A coordinate system is introduced in [2] and adopted in our algorithm. As showed in Fig.2, the lowest left corner has coordinates (0, 0), and each CLB has span of 1 on x axis and 1 on y axis, while each RAM or Multiplier has spans of 1 and 4, respectively.

We also adopt following definitions in this paper:

Definition 1. **Realization:** A realization r=(x, y, w, h) is a rectangle on FPGA fabric which can fulfill a module's resource requirement, where (x, y) specifies the lower left point of the rectangle, w and h are the width and height of the rectangle, respectively.

Definition 2.**Irreducible Realization**: Suppose r=(x, y, w, h) is a realization of module m, r is an irreducible realization *iff* there doesn't exist another realization r'=(x, y, w', h') with inequality $w' \leq w \wedge h' \leq h$ holding.

Definition 3. **Irreducible Realization List (IRL)**: The IRL of module m at point (x, y) is defined as the set of all the irreducible realizations of m at the same point, and denoted as IRL (m, x, y).

For example, assume resource requirement vector of a module m is (11, 1, 1). As showed in Fig.2, rectangle r_1=(5,0,5,4), r_2=(5,0,4,6), r_3=(5,0,3,11) and r_4=(5,0,5,5) are realizations of m at (5, 0) since the resources enclosed in them fulfill the resource requirement of m. It holds that IRL(m, 6, 0) is $\{r_1, r_2, r_3\}$, since only r_1, r_2 and r_3 are the irreducible realizations.

3. Sequence Pair

The proposed algorithm uses sequence pair to represent floorplans. A sequence pair $<\Gamma_+, \Gamma_->$ for a set of modules is two sequences of all the module names. The relative positions of the modules can be derived from a given sequence pair by the following two rules:

(1) (<...a...b...>, <...a...b...>)=>a is placed left to b;

(2) (<...b...a...>, <...a...b...>)=>a is placed below to b;

Several algorithms have been proposed to get the coordinates of each module [4, 8, 9]. All these algorithms calculate modules' x and y coordinates separately. Since the potential shapes of a module is relative to its coordinates, an evaluation algorithm is needed which can get a module's both x and y coordinate simultaneously. In this paper, we scan Γ_- from left to right. For each module obtained, all the modules left to or bottom to it can be found by scan Γ_+ from right to left. Thus, we can get modules' x and y coordinates simultaneously. The time-complexity of this algorithm is $O(n^2)$.

4. Proposed Algorithm

Since heterogeneous FPGA floorplanning is a seriously constrained problem, it is quite difficult for general simulated annealing algorithm to obtain feasible floorplans. Hence we present a new method called instance augmentation to control the stochastic optimization. Our experiments show that this method is quite fast and can achieve high success ratio. In this section, we first introduce the method of instance augmentation, then some details of the proposed algorithm.

4.1. Instance Augmentation

The main idea of instance augmentation is to use a way like branch-and-bound to prune beyond hope search and confine the solution space. Given a set of modules, it starts with a small instance composed of a subset of the modules. When feasible solution of current instance is found, it augments this instance to a bigger instance by inserting some left modules. Otherwise, it turns to a smaller instance by removing some modules of current instance. This process continues until feasible floorplan for the given instance found, or certain stop criteria met. Simulated annealing is used to find feasible solutions of each instance.

To make our illustration easier, we first introduce following denotations and definitions.

Given a set of modules M and a target device, we denote the floorplanning instance with all these modules as $I(M)$, and call $I(M')$ a sub-instance of $I(M)$, where M' is a subset of M.

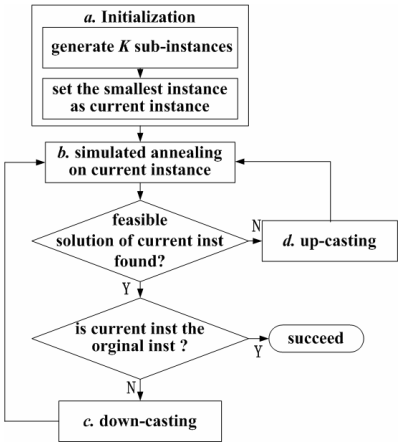

Figure 3. General flow of the proposed algorithm

Assume $S(M_i)$ is a solution of $I(M_i)$ ($M_i \subseteq M$), we call $S(M_i)$ a feasible solution *iff* all the modules in M_i are floorplanned in the target device without overlapping and their resource requirements are satisfied.

Definition 4. Projection Let $S(M)$ be a solution of $I(M)$. By removing all the modules not belong to M' from both sequences of $S(M)$, a sequence pair on M' is obtained. We call this new sequence pair *the projection on M' of S(M)*.

For example, (<*bdac*>,<*abdc*>) is the projection of (<*bdeac*>,<*abdce*>) on {*a,b,c,d*}, which is obtained by removing *e* from both of its sequences.

The general flow of instance augmentation is showed in Fig.3, and each key step will be introduced in the following.

a. Initialization:

The proposed algorithm first generates K subsets of M, which have $M_K \subset M_{K-1} \subset ... M_2 \subset M_1 \subset M_0 = M$. As stated in [11], modules with larger sizes have less flexibility in floorplanning. It's reasonable to place the modules with larger resource requirements first. When initializing, all the modules are weighted by their resource requirements. We select N-i weightiest modules into M_i, where N is the number of total modules and $0 \leq i \leq K$. When all the K subsets defined, we set $I(M_K)$ as the current instance and a solution of this instance, $S(M_K)$, is generated randomly as the initial solution. Note that in our experiments, K was set to large enough so that it's unlikely to fail to find a feasible solution of $I(M_K)$ after a search process.

b. Simulated Annealing on Current Instance:

The simulated annealing on current instance tries to find a feasible solution of this instance $I(M_i)$ ($0 \leq i \leq K$). It starts with $S(M_i)$, which is obtained from down-casting or up-casting or, for $I(M_K)$, randomly generated. The perturbations adopted are:

(1) swap two modules in one sequence;
(2) swap two modules in both sequences;
(3) change the current realization of a module;

Once $S(M_i)$ is feasible for the given outline, the search process terminates. Note that since each simulated annealing starts at a very low temperature and few repeated times are set for each temperature, a single search process is quite fast and the solution space it can explore is definitely confined.

c. Down-casting:

Once $S(M_i)$ gets feasible, the algorithm first generates $S(M_{i-1})$ by inserting m ($m \in M_{i-1}$-M_i) to both sequences of $S(M_i)$. We will try lots of different positions and shapes for m to get a better $S(M_{i-1})$. Since experiments show that many feasible solutions of the bigger instance can be obtained directly by inserting one module to a feasible solution of the sub-instance.

After that, the algorithm sets $I(M_{i-1})$ as the current instance and take $S(M_i)$ as the initial solution of the coming simulated annealing.

d. Up-casting:

If $S(M_i)$ is still infeasible after a simulated annealing, the algorithm keeps tracing back to smaller instances until the solution of the smaller instance is different from the projection of $S(M_i)$ or, the smallest instance is reached.

Then the algorithm sets the small instance as current instance and takes the projection as initial solution.

e. Stopping Criteria:

Once a feasible solution of the original instance is found, the algorithm succeeds and terminates; otherwise, when total iterations of down-casting and up-casting exceed a given threshold T_{term}, we regard the proposed algorithm as failed and terminate it.

4.2. Implementation Details

Following are some implementation details of the proposed algorithm.

a. Cost Function:

The cost function adopted in the proposed algorithm is:
$C = \sum \alpha |ratio\text{-}1| + \beta(ex\text{-}wire\text{-}length) + \gamma(ex\text{-}outline)$
where $\gamma >> \alpha >> \beta > 0$, *ratio* is the module's aspect ratio, and *ex-wire-length* is the total external wire length measured as center-to-center Manhattan distance between module pairs times the number of nets between them. *ex-outline* is the sum of the excessive width and height to the target device, i.e.
$ex\text{-}outline = max(H - H_d, 0)/H_d + max(W - W_d, 0)/W_d$
where W and H are the width and height of the floorplan, W_d and H_d are the width and height of the target device.

b. Calculation of IRL:

$O(W_d)$-time algorithm exists to calculate a module's IRL at a point. Note that to avoid wasting time in computing IRLs unnecessarily, $IRL(m, x, y)$ will not be calculated until module m is placed to (x, y) and the calculated IRLs will be preserved for future use. Since we prefer small aspect ratios of modules for the sake of wirelength optimization, we only consider those realizations with relatively small aspect ratios. This can greatly reduce the length of IRLs, thus improving the runtime.

c. Selection of Realization:

In our algorithm, we randomly choose the realizations of modules. Considering the internal wirelength and the fixed-outline requirement, we use a linear combination of realizations' aspect ratios and areas to decide their chance to be selected. The realizations with small aspect ratios and small areas have larger chance to be selected. We also adjust the weights of aspect ratio and area dynamically. When current instance is small, realizations with small aspect ratio have larger chance, while areas are more decisive when current instance is large.

5. Experiments

We implemented the proposed algorithm in C++, and run it on a PC with Intel® Celeron™ 2.4GHZ CPU and

512MB memory. We employed Xilinx's *XC3S5000* (the largest of Xilinx' *Spartan3* family) as target device, which has 80 columns of CLBs, 4 columns of interleaved RAMs and Multipliers.

We generated 5 group of testing data by recursively cutting the target device vertically or horizontally. The resource utilizations of these testing data range from 70% to 95%. Each group has 5 testing data of same number of modules and resource utilization. We run the proposed algorithm on each testing data for 20 times. Once a feasible solution is found, we deem the algorithm as succeeded, otherwise failed.

Since there are no commercial tools of heterogeneous FPGA floorplanning, we only compare our results with [2]. Table 1 shows the average run times of our algorithm and those of [2] grouped by the resource utilization of the testing data. Since no success rates are reported in [2], we only list ours. It can be seen that our algorithm succeeded in most cases in quite short time compared to [2]. We also generated a circuit which has 20 modules and resource utilization of 100%. This is a very tight problem and our floorplanner can get a feasible floorplan in 1.2 seconds, while for the same testing data, [2] needs 88 seconds. Fig.4 shows the resultant floorplan. Fig.5 shows a resultant floorplan of 50 modules with all three kinds of resource utilization of 90%, which is obtained in 2.7 seconds.

Table 1. Results of different testing data. [2] is carried out on a desktop with a 2.4GHz Intel® Xeon™ CPU.

# of modules	CLB rate	RAM rate	MUL rate	time(sec)		success rate
				ours	[2]	
21	72%	88%	86%	0.1	1	100%
23	83%	81%	81%	0.1	4	100%
37	94%	75%	75%	2.4	173	100%
50	78%	78%	76%	0.6	28	100%
100	78%	79%	77%	1.7	40	91%

6. Conclusion

Heterogeneous resources across the FPGA fabric have made FPGA floorplanning quite different from ASIC floorplanning. In this paper, we present a new algorithm addressed to this problem, which is based on a novel method called instance augmentation by us. Our algorithm is targeted to Xilinx *Spartan3* family and *Vertex-II* family, the most typical heterogeneous FPGA architecture. Experimental results based on Xilinx *XC3S5000* show that our algorithm is quite efficient and robust.

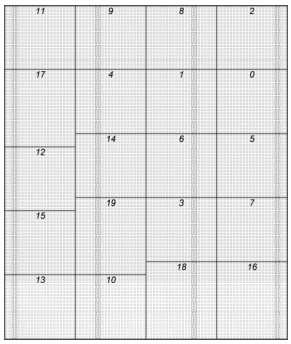

Figure 4. Floorplan of 20 modules

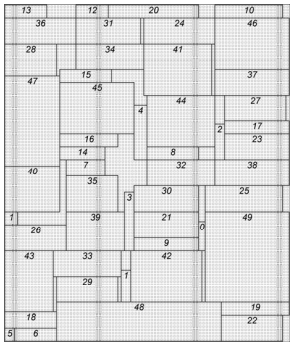

Figure 5. A resultant floorplan of 50 modules

7. References

[1] T. Taghavi, S. Ghiasi, A. Ranjan, S. Raje, M. Sarrafzadeh, "Innovate or Perish: FPGA Physical Design", *Proc. of ISPD*, pp. 148-155, 2004.

[2] Lei Cheng, Martin D. F. Wong, "Floorplan Design for Multi-Million Gate FPGAs", *Proc. of ICCAD*, pp. 292-299, 2004.

[3] D. F. Wong, C. L. Liu, "A new algorithm for floorplan design", *Proc. of DAC*, pp.101-107, 1986.

[4] H. Murata, K. Fujiyoshi, S. Nakatake and Y. Kajitani, "VLSI Module Placement Based on Rectangle-Packing by the Sequence Pair", *IEEE Trans. On CAD*, vol 15(12), pp. 1518-1524, 1996.

[5] S. Nakatake, K Fujiyoshi, H. Murata et al, "Module placement on BSG-structure and IC layout application", *Proc. of ICCAD*, pp.484-490, 1996.

[6] X. Hong, G. Huang, Y. Cai et al, "Corner block list: An effective and efficient topological representation of nonslicing floorplan", *Proc. of ICCAD*, pp. 8-12, 2000.

[7] Y.-C Chang, Y.-W. Chang, G.-M Wu, S.-W Wu, "B*-tree: A new representation for non-slicing floorplan", *Proc. of DAC*, pp.458-463, 2000.

[8] X. Tang, R. Tian and and D. F. Wong, "Fast Evaluation of Sequence Pair in Module Placement by Longest Common Subsequence Computation", *Proc. of DATE*, pp. 106-111, 2000.

[9] X. Tang and D. F. Wong, "FAST-SP: A Fast Algorithm for Module Placement Based on Sequence Pair", *Proc. of ASP-DAC*, pp. 521-526, 2001.

[10] A.B. Kahng, "Classical Floorplanning Harmful?" *Proc. of ISPD*, pp. 207-213, 2000.

[11] SheQin Dong, XianLong Hong, YuLiang Wu, Jun Gu, "Deterministic VLSI module placement algorithm using less flexibility first principle", *Journal of Computer Science and Technology (JCST)*, Vol. 18, No. 6, 2003.

[12] http://www.xilinx.com.