Fast Wirelength-driven Partition-based Placement for Island Style FPGAs*

Wentao Sui, Sheqin Dong, Jinian Bian, Xianlong Hong

Department of Computer Science and Technology, Tsinghua University, Beijing, China
suitwt05@mails.tsinghua.edu.cn, { dongsq, bianjn, hongxl }@mail.tsinghua.edu.cn

Abstract

In this paper, we propose a placement method for island-style FPGAs. This method consists of three steps: recursive bi-partition with terminal propagation consideration, minimum-cost flow initial placement and low temperature simulated annealing optimization. Unlike the traditional partitioning-based technique that is based on min-cut partitioning, we apply ratio partitioning in each level. For each partitioning region, minimum-cost flow algorithm is used to determine the initial placement. We use low temperature simulated annealing to improve the initial placement result. Experimental results show the efficiency and effectiveness of our algorithm.

Keywords: Placement, Partition, Minimum-cost Flow, SA

1. Introduction

Placement of FPGA is the physical design phase in which a netlist of circuit blocks is mapped onto physical locations typically arranged in a two dimensional array.

Most of the recent CAD algorithms usually take hours to map, place and route circuits with millions of gates on a state-of-the-art FPGA chip. This may nullify its time-to-market and in particular, the advantages of reconfigurability. With increasing emphasis on reconfigurable computing, there is a pressing need for very fast CAD tools without sacrificing the quality of solution.

The three major classes of placers used today are partitioning-based, analytic-based which are often followed by local iterative improvement, and simulated annealing (SA) based placers. Although simulated annealing algorithms produce good results, it requires enormous computation time which may depend on the initial configuration of the placement. Traditionally, partition-based placement algorithms [1, 2] have been fast and hence scalable for larger design of the future. Therefore, we shall develop our placement algorithm based on the partitioning-based approach.

In this paper, we present a wirelength driven partitioning-based placement algorithm. The placement algorithm consists of three main steps: logic function blocks placement, IO blocks placement and placement optimization (Fig. 1). Unlike the traditional partitioning-based technique that is based on balanced partitioning, we partition the net according the net-weight which calculated dynamically. The net weight varies according terminal propagation during partitioning progress. We use the minimum-cost flow to determine the initial placement, the edge weight in the flow graph is related with partition stage and physical position on the chip. To improve the initial placement result, a low temperature SA is used.

The rest of the paper is organized as follows: Section 2 presents our partition-

*This work was supported by National Natural Science Foundation of China under grant NSFC-90607001

© the authors

based placement algorithm. In Section 3, simulation results are shown.

2. Proposed algorithm

To improve the solution quality and reduce the time of placement, we combine the partitioning-based and the SA based algorithms together.

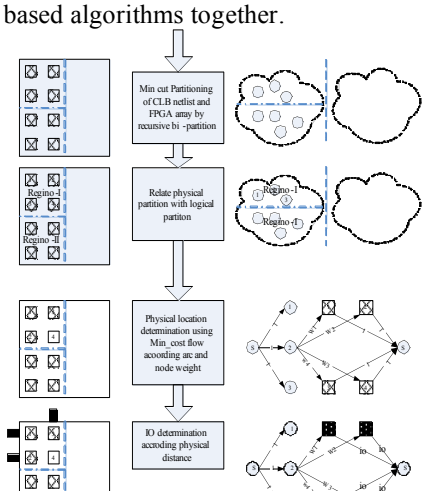

Fig. 1: algorithm flow.

2.1. partition-based placement

In constructive placement approaches, a circuit is either recursively bisected in a top-down fashion [4] or basic elements are clustered in a down-top fashion [3].

We model the technology-mapped CLB netlist as a hyper-graph *H(V, E)*. Each vertex in the set *V* corresponds to a logic block in the netlist. Each hyper-edge in the set *E* represents a net in the logic connection.

The hypergraph is bi-partitioned recursively using the state-of-art hyper-graph partition tool hMetis [6]. HMetis reduces the size of the graph by collapsing vertices and edges (coarsening phase), partitions the reduced graph (initial partitioning phase), and then uncoarsens it to construct a bi-partition for the original graph (uncoarsening and refinement phase).

During the logical partition progress, the physical chip is partitioned vertically and horizontally at the same time. When the number of blocks in each partition equals to or is less than some predefined threshold value, the partitioning process ends.

2.2. net-weight computation

In [7], a hierarchical ratio partitioning method was presented. We also employ the net-weight estimation method similar with [7].

The logical vertexes can only be mapped onto the dispersed position with CLB on the FPGA. We mark the CLB coordinates with integer pair *(x, y)*, this is distinguished with the coordinates on the ASIC which the function blocks can be mapped onto continuous position of the chip. So the net-weight estimation is fast. There are three situations which we should update the weight (Fig. 2).

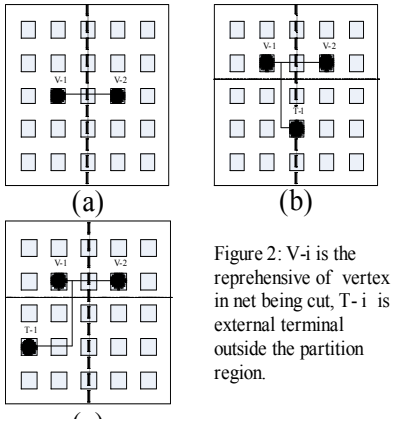

Figure 2: V-i is the reprehensive of vertex in net being cut, T- i is external terminal outside the partition region.

Fig. 2: terminal consideration.

Before partition, we first allocate all logical vertexes in the center of the FPGA. Each hyperedge corresponds to a multi-terminal net in the circuit. At first each net is assigned a weight of half width of the chip if the first cut direction is vertical. In *Fig. 2*, the *V-i* represent the vertex of the circuit being cut, the *T-i* represent the vertex outside the partition regions which have connection with

© the authors

netlist currently being cut. In *Fig. 2(a)*, when the net has only two vertexes, the net weight is set to the half region width, for the reason that if the net is cut between the two vertex, the wirelength contribution got by these two vertex is half of the partitioning region width. In *Fig. 2(b)*, we add the net weight with *dist(v2, t1) – dist(v1, t1)*. In this case, if the net doesn't be cut, the HPWL contribution got from the net is *dist(v1, t1)*, this is certain. When the net is cut, the HPWL increase is *dist(v2, t1)*, then we should decrease the dist*(v1, t1)* from *dist(v2, t1)*. In *Fig. 2(c)*, the terminals are outside the horizontal range of *V-1* and *V-2* and closer to *V-1*, the hyperedge weight is appended with *dist(v1, v2)*. Each time the partitioning ends, if the net isn't being cut, the net weight value is restored as prior to partitioning.

In order to compute the hyperedge HWPL efficiently, we use an array to recorder the left-bottom and top-right bounding-box coordinates of each net. After each partitioning, if the corner coordinate varies, the bounding-box coordinates of the net are updated.

2.3. min-cost flow placement

When the vertexes number in the current partition below to some threshold value, the net-weight based partition ends and min-cost flow placement begins. There are two kinds of vertexes in the net-flow paragraph (*Fig. 3*), the logical vertexes and the physical vertexes. The logical vertexes are corresponding with blocks in the circuit and the physical vertexes are corresponding with CLBs on the chip one by one. We add a Start and a Terminate vertex to the net flow graph. There are three categories edges: edges form start node to logical vertexes, edges from logical vertexes to physical vertexes and edges form physical vertexes to the end vertex.

The weights and flow constraints of edges from S node to logical vertex and from physical vertex to *T* node are all set with 1. According to the max flow constraint conditions, the outflow of *S* and the inflow of *T* are set to the logic vertex number, which means to accommodate logical vertexes onto physical vertexes.

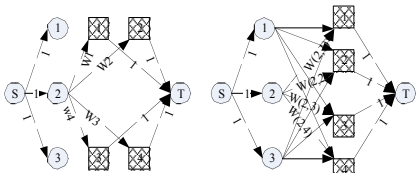

Fig. 3: max flow graph

What is important is the edge between the logical and physical vertexes. Each physical vertex is different important to the same logical vertexes (*Fig. 4*). In *Fig. 4,* when we assign *V-3* to different physical location in the current partition region, we can get different HPWL, so we set different weight to different physical location. To different net, the same physical position has different weight too. We add the edge weight and physical vertexes weight together on the edges between logical vertexes and physical vertexes in the flow graph.

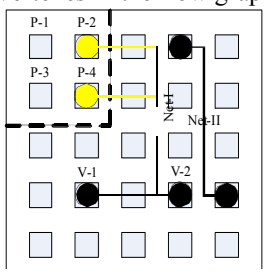

Fig. 4: physical vertex weight

We solved the network flow formulation using a software package developed by Goldberg [9].

2.4. Placement of IOBs

We place the IO blocks using the min-cost flow algorithm too. But the capacity of edges from physical nodes to terminal node is set to io-rate, the num-

© the authors

ber of IO pins in each CLB column and row in FPGA.

2.5. Low temperature SA

To improve the quality of placement further, an ultra-low temperature SA is executed on it to obtain the final placement configuration. Through experiment, we deduced the starting temperature of the cooling schedule to be 0.005σ, where σ is the standard deviation of the BB-costs of applying n random swaps of adjacent blocks (the number of CLBs being n).

3. Experimental result

Table 1 show the characteristics of nine MCNC benchmark circuits along with the results obtained by our placement method. We compare our placement result with the result got from VPR, a famous FPGA placement tool.

Form the column 8, we can see that our initial placement result are 24.5% better than initial placement of VPR in average. Although the final bounding-box cost of ours and the VPR are similar, the runtime of our placement is 32.4% faster than VPR's, which shows in column 15.

4. References

[1] U D. J.-H. Huang and A. B. Kahng, "Partitioning-based Standard-cell Global Placement with an Exact Objective", *Proc. ACM/IEEE ISPD*, 1997.

[2] M. Wang, X. Yang and M. Sarrafzadeh, "DRAGON2000: Standard-Cell Placment Tool for Large Indeustry Circuits", *Proc. ACM/IEEE ICCAD*, 2000.

[3] P. Banerjee, S. Bhattacharjee, S. Sur-Kolay, S. Das, S. C. Nandy, "Fast FPGA placement using Space-filling Curve", *Proc. Intl. Conf. Field Programmable Logic and Application*, 2005.

[4] P. Banerjee and S. Sur-Kolay, "Faster Placer for Island-style FPGAs", *Proc. International Conference on Computing: Theory and Application*, 2007.

[5] A. E. Dunlop and B. W. Kernighan, "A Procedure for Placement of Standard-cell VLSI Circuits", *IEEE Transactions on Computer-Aided Design*, 1985.

[6] http://www-sers.cs.umn.edu/karypis/metis/hmetis/

[7] T. C. Chen, T. C. Hsu, "NTUplace: A Ratio Partitioning Based Placement Algorithm for Large-Scale Mixed-Size Designs", *ISPD*, 2005.

[8] V. Betz, and J. Rose, "VPR: A New Packing, placement and routing Tool for FPGA Research," *in 7th Intl. Workshop on Field-Programmable Logic and Applications*, 213-222, 1997.

[9] A. V. Goldberg, "An efficient implementation of a scaling minimum-cost flow algorithm," *J. Algorithms*, 1997.

Table 1. Benchmark details, and comparison of cost and time: our method vs. VPR

Circuit	block Num	IO Block	Net Num	chip dimension	Initial bb-cost			Final bb-cost			Runtime time(s)			
					Vpr	Our	Improve(%)	Vpr	Our	Diff	initial	final	Vpr	Improve(%)
Apex2	526	41	1405	40x40	582.984	397.748	**31.8**	184.676	181.775	-2.901	3.4	16	25	**36**
Apex4	362	28	959	19x19	205.753	160.67	**21.9**	121.014	121.207	0.193	1.3	9	16	**43.75**
bigkey	853	426	1037	27x27	329.8	230.907	**30**	132.369	130.91	-1.459	2.5	20	34	**41.2**
Clma	2280	144	6128	47x47	2876.31	2800.87	**3**	1002.22	1001.85	-0.37	42.1	204	281	**27.4**
des	916	501	1503	32x32	497.731	320.729	**35**	160.652	158.014	-2.638	4.0	25	32	**21.9**
diffeq	482	103	1180	20x20	241.475	207.801	**13.9**	99.8887	100.687	0.7983	1.0	14	20	**30**
dsip	769	462	920	27x27	298.163	203.965	**31.6**	117.288	118.317	1.029	2.7	20	26	**23.1**
elliptic	1151	245	2450	31x31	752.888	580.124	**22.9**	342.459	347.669	5.21	6.6	41	71	**42.3**
S38584	1955	342	4706	41x41	1910.36	1330.83	**30.3**	530.87	529.624	-1.246	15.7	120	162	**25.9**
Avg:							**24.5**							**32.4**

© the authors