A Novel Incremental Algorithm for Non-slicing Floorplan with Low Time Complexity*

Liu Yang[1], Sheqin Dong[1], Xianlong Hong[1], Yuchun Ma[1]
EDA Lab., Dept. of Computer Science & Technology, Tsinghua University, Beijing 100084, China
yliu02@mails.tsinghua.edu.cn

Abstract

Incremental modification and optimization is very important in VLSI physical design. In this paper, we present a novel incremental algorithm for non-slicing floorplan based on Corner Block List representation with low time complexity. To implement incremental floorplan, constraint graphs are produced simultaneously with the packing process of transform CBL to its corresponding floorplan, which takes O(n) time. Based on the critical path and the accumulated slack distances, we can determine the balance point of insertion and do a series of operations incrementally, such as deleting, inserting, resizing modules quickly. And the time complexity of incremental modification process is O(n). The experimental results show that wirelength of all incremental modifications are decrease compared with the initial floorplan. The total wirelength of all incremental floorplans have been improved from 0.1% to 9.0%. And the CPU time of incremental modifications are so small that most of them are within micro second. The experimental results of MCNC benchmarks prove the effectiveness of our algorithm.

Keywords: incremental floorplanning, constraint graph

1. Introduction

Traditionally, CAD tools have been used more or less independently at the system-level, high-level, logic-level, and layout-level, each with its own set of synthesis, verification, and analysis tool-set. Each tool is unaware of the big picture and the mechanism linking them together. But with the continuous shrinking of feature size, deep submicron effects cause more and more interactions between high-level and physical-level design. Recently, several synthesis researches have been reported taking physical information into account, such as [1]. Complexity of present and next generation VLSI system complicated with iterative processes as faced in many complex projects requires very efficient incremental design algorithms and methodologies. At the same time, time-to-market pressure is driving the electronic design automation strategies to reconsider design methodologies. It is no longer affordable to redo the time-consuming process of floorplanning and associated top-level routing after each of these changes. Hence, we should take new approaches to these problems.

More and more people are interested in the research on the interactions between high-level and physical-level design. To cope with the complexity of the merging of those design phases, incremental algorithms are becoming more and more important. Floorplanning is an important stage in the physical design cycle, it can provide necessary information to estimate quality metrics, such as area, wire length, net capacitance (which is based on wire length). These physical metrics are important estimations for high-level SOC synthesis[14], and are becoming more and more popular in SOC timing closure design in the future. Thus, researches on the incremental floorplanning algorithms are relevant to the interactions between floorplan and high-level synthesis.

There are some related research papers in the area of incremental algorithms. In [2], Cong et al. formulated incremental physical design problems and surveyed the existing solutions. Crenshaw et al. [3] proposed an incremental floorplanner which used a greedy method to apply local changes on a slicing floorplan tree. The algorithm was applied to high-level synthesis framework. It used an area-only criteria to assess the need for invoking a traditional floorplanner. Liu et al.[4] devised an incremental floorplanner based on genetic algorithms. It supported incremental changes on an existing floorplan and it can find an acceptable solution with a magnitude of speedup compared with traditional approaches.

The major contributions of this paper are: A novel incremental algorithm for non-slicing floorplan based on corner block list representation is proposed. To implement incremental floorplan operations, constraint graphs are produced simultaneously with the packing process of transform CBL to its corresponding floorplan. The time complexity of the transformation is O(n) compared with the method of scanning line. In our algorithm, .The process of finding the balance node and inserting the adding modules only takes O(n) time, where n is the number of the floorplan modules. The CPU time of our incremental algorithm is within micro second.

The rest of the paper is organized as following. In Section 2, we formulate the incremental floorplan problem and introduce the flowchart of our algorithm. Then in Section 3, we describe our incremental floorplanning algorithm in detail, especially the operations based on constraint graphs. In Section 4, some experimental results are given. Finally, we conclude our work in Section 5.

2. Problem Formulations

2.1 Flowchart of System Integrated Synthesis and Floorplanning

There are a lot of research works on creating a system integrated synthesis and floorplanning in universities and in industries. During the design process, synthesis tool provides an initial full floorplan which is created from a baseline netlists. During the next phase, as new design automation decisions are evaluated, updates are made incrementally on the floorplan. The packing results can be used as an indication of whether the incremental updates were sufficient (if it is a significantly better solution, then a better update threshold should be used). Fig.1 shows the flowchart of system integrated synthesis and floorplanning.

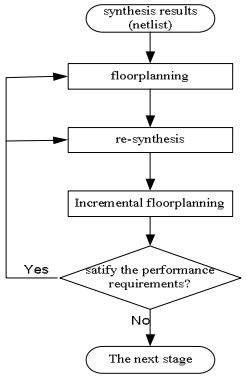

Fig.1 Flowchart of System Integrated Synthesis and Floorplanning

In the phase of high-level synthesis, there are different binding

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

solutions which lead to the changes of floorplanning configurations. The operations in synthesis include the binding moves of share, split and swap[5].

The operation of "Share" merges two resources res1 and res2 into one single resource. The resulting impact on a floorplan is that the corresponding initial two blocks merge into one block. Similarly, the operation of "Split" is the reverse of the operation of "share". A single resource is split into two resources. There is also another operation of "Swap". "Swap" occurs when the function of some module can be replaced by the other module which may have better performance than the original one. The corresponding operations on floorplan arise from three instances described above are "Deleting Modules", "Inserting Modules" and "Resizing Modules".

2.2 Incremental Floorplanning Problem

After the synthesis or floorplan phase (sometimes even after the routing phase), we would like to move some modules which have been packed well or add some new modules in the chip, which often results in overlaps among neighboring modules or an infeasible floorplan (some modules' location exceed the borders of the chip). Therefore, we need to create an efficient mechanism to ensure that we can finally acquire a new feasible floorplan (satisfies fixed area constraint and with no overlaps) which is as close to the former infeasible floorplan as possible. The incremental floorplan phase addresses this problem.

So our problem can be described as follows: Given an initial floorplan result F based on CBL and the modification request of the synthesis. Based on given binding moves, we modify floorplan F locally into F' while we change the modules' topological relations as less as possible. As a result, we have kept the good performance derived by the original packing and on the other hand we can simultaneously get the CBL list of the modified floorplan which can be used in the next optimizations.

The objective driving the floorplanner is to minimize the area and wire length of whole chip and the number of those blocks which need movement(for short: N_{mb}).

3. Incremental Floorplanning Algorithms

3.1 CBL Representation

As an effective representation for non-slicing floorplanning, CBL[6] has advantages in both time complexity and solution space over some other representations. So in our algorithm, we adopt the representation of CBL to represent the floorplan. The most important is that by using CBL representation, it only takes O(n) time to transform CBL to a corresponding floorplan when the total number of blocks is n. This will be discussed in the next section.

3.2 Construct Constraint Graph with Low Time Complexity

Compared with the method of scanning line, whose graph-built-up process by scanning the coordinates of all modules one by one which takes the computation complexity of $O(n\log_2 n)$, the process of building up constraint graphs in our algorithm is embedded in the process of packing and the computation complexity of the transform from a given CBL to a corresponding floorplan remains O(n). This is the reason why we adopt CBL presentation. The corresponding algorithm is shown below:

Algorithm 1. Transformation from corner block list to constraint graph

INPUT: A corner block list of a floorplan, i.e. a triple list of (S,T,L)

OUTPUT: horizontal constraint graph(G_h) and vertical constraint graph(G_v) of the floorplan

BEGIN

(1) Build G_h and G_v including all nodes representing all blocks;

(2) Initialize the floorplan with block S[1]; Add the edge of connecting s and module_id=S [1] in G_h and G_v;

(3) **FOR** i = 2 to n **DO**

IF L[i]=1(module orientation is horizontal) **THEN**

Insert block S[i] with an orientation L[i] and cover the T-junctions according to the record in list T;

Add the edge of connecting the current module and his low covered modules to G_v;

Add the edge of connecting the current module and his left modules to G_h;

ELSE IF L[i]=0(module orientation is vertical) **THEN**

Insert block S[i] with an orientation L[i] and cover the T-junctions according to the record in list T;

Add the edge of connecting the current module and his left covered modules to G_h;

Add the edge of connecting the current module and his low modules to G_v;

ENDFOR

(4) Add the edge of connecting the right boundary modules and the ending modules to HCG;

Add the edge of connecting the upper boundary modules and the ending modules to VCG;

END.

Theorem 1: The time complexity of algorithm 1 is O(n).

Proof: We have the conclusion that the length of list T is no more than 2n-3 in [7]. The time complexity of transformation from CBL to floorplan is as the same as the length of list T, so it is O(n); and from the algorithm above, we can obviously see that the time complexity of algorithm 1 is as the same as the time complexity of packing process. Therefore, the time complexity of algorithm 1 is O(n).

We build up two directed acyclic graphs which record the topological relations between the blocks in the packing of floorplan: the horizontal constraint graph G_h and the vertical constraint graph G_v (see Fig.2 and Fig.3).

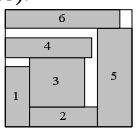

Fig.2 An example of the packing of the floorplan on CBL

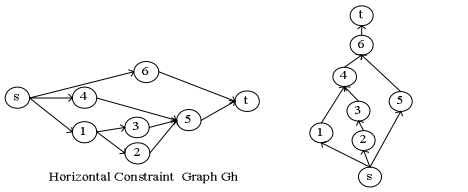

Fig.3 The constraint graphs of the packing of the floorplan on CBL

We set $G_h=(V_h, E_h)$, and $V_h=\{s_h, v_1, v_2, v_3, \ldots, v_n, t_h\}$, where s_h is the source node in G_h, t_h is the target node in G_h, v_i represent block B_i which record the information of width, height, room position et.al. $E_h=E_1 \cup E_2 \cup E_3$, where $E_1=\{e(i, j) \mid$ "block i and j are adjacent" & "block i is on the left of block j"$\}$, $E_2=\{e(i, j) \mid$ "an edge from s_h to each of the vertices representing the leftmost blocks."$\}$, $E_3=\{e(i, j) \mid$ "an edge from each of the vertices representing the rightmost blocks to t_h."$\}$. And $G_v=(V_v, E_v)$ can be defined similarly. In zG_v and G_h, ***d(i, j)*** denotes weight of ***e(i, j),*** that is the slack distance between the corresponding block i and block j.

3.3 Some Definitions in Constraint Graph

Based on constraint graphs, we have:

Definition 1 [Critical Path (CP)] : *In each constraint graph, there is a critical path which has property: CP={e(i, j) | $\forall$ e(i, j) $\in$ CP, there must have d(i, j) =0 }. That is to say, the slack distance of any arbitrary two blocks corresponding to the two nodes on the critical path is zero.*

Lemma 1: *In each constraint graph, there must be one or more than one critical path, among which at least one critical path determines the dimension of the floorplanning packing in its corresponding direction.*

Definition 2 [Critical Node(CN)] *For each block B_i, there must have a critical node in G_h and G_v which is the preceding node connecting to v_i in CP_h and CP_v respectively.*

Lemma 2: *For each block B_i, there must have a CN_i in each direction and each CN_i determines the position of B_i in the corresponding dimension.*

For example in Fig.3, CP_h is the path which consists of v_1, v_2, v_5, and $CN_h(v_5)$ is v_2.

For any node $v_i \in G_h$, we also define ***the slack of v_i*** as the accumulated slack distances in all paths from v_i to the target t_h.

The slack of v_i in G_h is given by:

$$sl_x(i) = \begin{cases} d(i,k) + \min\{sl_x(k)\}, & \text{if } set != null; \\ d(i,k), & \text{if } set = null; \end{cases} \quad (1)$$

Where ***k*** $\in$ ***set, set***=$\{$the successors of node i in $G_h\}$; ***d(i,k)*** is the weight of the edge ***e(i, k).***

Similarly, the slack of v_i in G_v is given by:

$$sl_y(i) = \begin{cases} d(i,k) + \min\{sl_y(k)\}, & \text{if } set != null; \\ d(i,k), & \text{if } set = null; \end{cases} \quad (2)$$

Where $k \in set$, set={the successors of node i in G_v};

When the adding module is inserted, there have two directions to be inserted. We do it in this way in order to keeping original topological nature of CBL representation. As we know, the Corner Block(CB) is the key of CBL representation, and the insertion directions of CB are two: from upper direction and from right direction. Therefore, we distinguish these two kinds of situations.

One way is inserting the adding module to the original floorplan from the direction of "right". In this case, the sl' of Vi in x and y directions are sl_x' and sl_y' respectively:

$$\begin{cases} sl_x' = sl_x; \\ sl_y' = sl_y + BlockH; \end{cases} \quad (3)$$

The equivalent expression is:

$$\begin{cases} sl_x' = d(i,k) + \min\{sl_x(k)\}; \\ sl_y' = d(i,k) + \min\{sl_y(k)\} + BlockH; \end{cases} \quad (4)$$

The other way is inserting the adding module to the original floorplan from the direction of "upper". In this case, the sl' of Vi in x and y directions are sl_x' and sl_y' respectively:

$$\begin{cases} sl_x' = sl_x + BlockW; \\ sl_y' = sl_y; \end{cases} \quad (5)$$

The equivalent expression is:

$$\begin{cases} sl_x' = d(i,k) + \min\{sl_x(k)\} + BlockW; \\ sl_y' = d(i,k) + \min\{sl_y(k)\}; \end{cases} \quad (6)$$

Where, BlockW and BlockH are the width and height of the adding modules respectively.

3.3 Incremental Floorplanning Operations

3.3.1 Deleting Modules

In "deleting" process, we consider two factors: one is decrease of total chip area, the other is total wirelength.

First, the factor of area is considered. Assuming the deleted node is v_i, whether the succeeding nodes are influenced depends on the critical node belonging to v_i by virtue of Lemma 2. For example, in Fig.3, v_2 is the critical node of v_5, whereas v_3 is not the critical node of v_5. Therefore, when we delete block 2, block 5 should be moved close to the rightmost one among all connected adjacency blocks on the left of block 5. But when we delete block 3, block 5 and all of its succeeding adjacency blocks need no movement. The rest may be deduced by recursive method. In this recursive process, the following criterion should be obeyed:

Criterion 1: *If and only if the deleting module lies in the critical path which is the only critical path including the deleting one in one direction of the whole chip, we need move those succeeding adjacency modules on the right/upper of the current deleting modules.*

This is obvious. But on the other hand, we should also consider that perhaps sometimes total wirelength is increasing on the contrary if the whole chip is compacted. So we take wirelength information into account at the same time. The cost function is:

$$C = \alpha * A + \beta * L \quad (7)$$

Where, the first item stands for chip area. The smaller the area is, the more superior the deleting process is. And the second item stands for total wirelength.

However, the two items are not coincident in many cases, so that determining which one is more important is critical. In our approach, we introduce α and β, called "*weight conciliation factor*", to solve the problem. The larger α or β is, the more importance the item gets.

3.3.2 Inserting Modules

The objective function of operation "inserting modules" is minimizing the increment of area and N_{mb}. The increment of area is given by:

$$\Delta area = (h + \Delta h) * (w + \Delta w) - h * w \quad (8)$$

Definition [balance node(BN)]: *"balance node" is the node BN. If the adding module is inserted above or on the right of BN, the increment of chip area is minimal.*

So the question is focused on how "*balance node"* is to be found. We assume W_p and H_p as width and height of the inserting module B_p respectively. Every node has the increment value of whole chip width and height which can be given respectively by two cases, if it is considered to be a chosen balance node.

One is that if the adding module is inserted on the right of BN, then we have:

$$\begin{cases} \Delta w = \max(0, (W_p - sl_x(i))) \\ \Delta h = \max(0, (H_p - (sl_y(i) + BlockH))) \end{cases} \quad (9)$$

Where, i is chosen balance node, W_p and H_p are the width and height of the inserting module B_p respectively.

And the corresponding increment area value is:

$$\Delta area = [h + \max(0, (H_p - (sl_y(i) + BlockH)))] * [w + \max(0, (W_p - sl_x(i)))] - h * w \quad (10)$$

The other is that if the adding module is inserted on the right of BN, then we have:

$$\begin{cases} \Delta w = \max(0, (W_p - (sl_x(i) + BlockW))) \\ \Delta h = \max(0, (H_p - sl_y(i))) \end{cases} \quad (11)$$

And the corresponding increment area value is:

$$\Delta area = [h + \max(0, (H_p - sl_y(i)))] * [w + \max(0, (W_p - (sl_x(i) + BlockW)))] - h * w \quad (12)$$

Initially, we calculate the slack value of every node in each direction by using the formula (9) and (11). we compare all blocks in reverse order of "Sequence" series in CBL. Every original dead space should be searched. If there is a dead space where the adding module can be placed close, we need do no other operations on finding balance node. All of the original blocks don't need any movement. But if a dead space can not be found, we will continue to compare the slack value of every node in the constraint graph. More detailed information is given by the following paragraphs.

We compare increment area which is derived from insertion of adding module until a balance node is found.

The pseudo-code about the operation of "inserting module" is shown as algorithm 2:

Algorithm 2. "Inserting modules" procedure

*(1) Let **Ba** is the adding module.*
*(2) Search in reverse order of list S in CBL. Let **Bi** is the current block which is being considered.*
*(3) **While** (Bi != null)*
* **if** Ba is inserted on the right of Bi*
* **if** the own slack space of Bi can be large enough to contain Ba*
* balance_node = Bi; record the insertion direction; break;*
* **if** Ba is inserted on the upper of Bi*
* **if** the own slack space of Bi can be large enough to contain Ba*
* balance_node = Bi; record the insertion direction; break;*
* Calculate incre_area_right and incre_area_above espectively;*
* Compare the incre_area values and update the minimal incre_area value.*
* **EndWhile;***
(4) Select Bi which has the maximal value of $\Delta area(i)$ as "balance node";
(5) Inserting the given adding module to the original packing of flooplanning and update those influenced modules' position;
(6) Update the sequence series of (S, T, L);
(7) Update the whole chip height and width and output the area.

Theorem 2: The time complexity of algorithm 2 is O(n).

Proof: The process of finding balance node is based on searching in reverse order of list S in CBL which has n blocks. For every node, when we calculate $\Delta area$, the minimal $\Delta area(i)$ and the current balance node i is memorized simultaneously, it is unnecessary to sort $\Delta area$ values of all blocks additionally. Therefore, the time complexity of algorithm 2 depends on n, which is the length of S. So the time complexity is O(n).

3.3.3 Resizing Modules

This operation can be divided to two stages: firstly, the corresponding original module is removed from the packing of floorplanning, i.e., the original node is deleted from two graphs; secondly, we select a insert node position by the method mentioned in chapter 3.2.2, and then add the swap module into it. This operation can be regarded as combination of deleting modules and inserting modules.

4. Experimental Results

We have implemented the incremental floorplanning algorithm in C programming language, and all experiments are performed on a SUN sparc20 workstation. Some MCNC benchmarks are used in the experiments.

We design the experiments by the method as this: Firstly, we delete several arbitrary blocks from the initial floorplan by our incremental algorithm on "deleting modules"; then we insert the same ones which had been deleted by our incremental algorithm on "inserting modules". We do in this way in order that we can get the original information about netlist and module size which facilitates us to calculate the whole wirelength.

Due to O(n) time complexity of our incremental algorithm, the CPU time of incremental modification is so fast that it is within micro second. This advantage will become more obvious with the increase of problem size.

Table 1 compares the floorplan results based on incremental algorithm and one-shot algorithm. The column of "operation" indicates the operation mentioned in this paper("(5,6,8)" means that firstly we delete block 5,6,8 in the initial floorplan, and then we insert new block 5',6',8' to the after-deleting floorplan, and new block 5',6',8' has the same netlist and module size information of delete block 5,6,8.

From the table we can see that wirelength of all cases after incremental modification is decreased compared to the initial floorplan. the total wirelength has been improved in 0.1% to 9.0%. The area and area usage are changeless at least in most cases and even reduced in example Hp.

We also give the packing result of one example Ami33 in Figure 4. Fig.4(a) is the initial floorplan; Fig.4(b) is the incremental algorithm result after the operation (3,17,24), in which purple blocks are the ones which are influenced and reshuffled in our incremental approach and the red block is the new inserted ones (34,35,36) which is the replication of (3,17,24). The detailed data of area and wire length in initial floorplan and incremental algorithm can be seen in Table 1. We can see that the area changeless and the wire length is improved by 0.4-0.7%, and only block 9, 23, 32 have been influenced and moved, and the total number of moved blocks is only 2 which is 9.1 percent of the initial ones.

5. Conclusion

In this paper, we present a novel incremental algorithm for non-slicing floorplan based on corner block list representation with low time complexity. The horizontal and vertical constraint graphs are built based on the packing of the initial floorplanning result, which only takes O(n) time complexity. Based on the critical path and the accumulated slack distances which are defined by us, we can determine the balance node of insertion and do a series of implement kinds of operations incrementally, such as deleting modules, inserting modules and, resizing modules quickly. And the time complexity of incremental modification process is O(n). Good performance of experimental results in dealing with instances taken from industry proves the effectiveness of our algorithm. We show that the incremental approach to floorplanning can be fast and still supply other tools with good physical estimates for area and wire length. The experiment results show that our algorithm is promising.

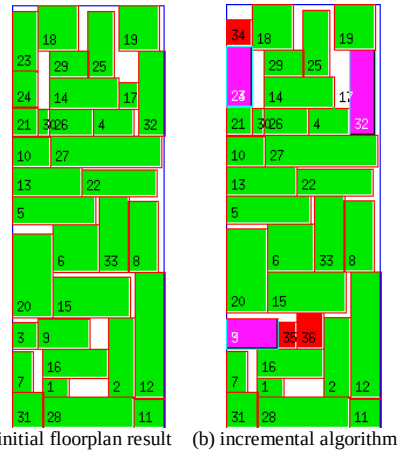

Fig.4 (a) initial floorplan result (b) incremental algorithm result

6. Reference

[1] A. Stammennann, D. Helms, M. Schulte, "binding, allocation and floorplanning in low power high-level", ICCAD'03. Novembcr 11-13,2003, San Lose, California, USA

[2] Jason Cong, Majid Sarrafzadeh. "Incremental Physical Design", in *International Symposium on Physical Design 2000.*

[3] Jim Creshaw, et.al. "An incremental floorplanner", Proceeding of IEEE, Great Lakes Sym. on VLSI, p248-251 1999

[4] Yongpan Liu; Huazhong Yang; Rong Luo; "An incremental floorplanner based on genetic algorithm", *ASIC,2003. Proceedings. 5th International Conference on*, Volume: 1 , 21-24 Oct. 2003 Pages:331 - 334 Vol.1

[5] A. Stammennann, D. Helms, M. Schulte, "Binding, allocation and floorplanning in low power high-level", ICCAD'03. November 11-13,2003, San Lose, California, USA

[6] Hong Xianlong, Huang Gang et al. "Corner Block List: An Effective and Efficient Topological Representation of Non-slicing Floorplan" *ICCAD'*2000.

[7] Hong, X.; Sheqin Dong; Gang Huang; Cai, Y.; Chung-Kuan Cheng; Jun Gu: "Corner block list representation and its application to floorplan optimization", Circuits and Systems II: Express Briefs, IEEE Transactions on [see also Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on] , Volume: 51 , Issue: 5 , May 2004, Pages:228 - 233

Table 1. The results of initial floorplan and incremental floorplan

Circuits	Instance Number	Operation	Area(mm^2)		Wirelength(um)			Area usage	
			F	F'	F	F'	**Improved**	F	F'
apte	9	(5, 6, 8)	49.178208	49.178208	756170	687832	9.0%	94.7%	94.7%
apte	9	(2, 3, 9)	49.178208	49.178208	756170	735770	2.7%	94.7%	94.7%
Hp	11	(9, 10, 11)	9.836064	9.680832	348424	339337	2.6%	89.8%	91.2%
Hp	11	(4, 5, 9)	9.836064	9.680832	348424	347625	0.2%	89.8%	91.2%
Ami33	33	(1, 17, 30)	1.296540	1.296540	127530	127040	0.4%	89.2%	89.2%
Ami33	33	(3, 17, 24)	1.296540	1.296540	127530	126613	0.7%	89.2%	89.2%
Ami49	49	(1, 8, 37)	41.160000	41.160000	1510590	1509610	0.1%	86.1%	86.1%
Ami49	49	(9, 40, 43)	41.160000	41.160000	1510590	1490529	1.3%	86.1%	86.1%

F stands for the initial floorplan results, *F'* stands for the incremental floorplan results.