Hierarchical Floorplan Design on the Internet

Jiann-Horng Lin *Jing-Yang Jou* *Hui-Ru Jiang*

Department of Electronics Engineering, National Chiao Tung University, Hsinchu, Taiwan

ABSTRACT

With the proliferation of transistor count in VLSI design, more and more design groups try to figure out a way to efficiently combine their designs. The Internet features distributed computing and resource sharing. Consequently, a hierarchical floorplan design can be adequately solved in the Internet environment. In this paper, we address the problem of area minimization floorplan design in the Internet environment. We propose a novel algorithm, RMG algorithm. Taking advantage of the Internet, RMG algorithm reduces the computing time by shortening the critical path in the floorplan tree. With creating floorplan design in the Internet environment, it can be seen that the Internet advantages Electronic Design Automation (EDA).

1. INTRODUCTION

Floorplan design is a stage in VLSI design which determines the locations of modules. Being applied before placement and routing, floorplan design dominates the resulting area and performance. The objectives of floorplan design vary from (1) to minimize area, (2) to reduce wirelength, (3) to maximize routability, (4) to minimize power consumption, (5) to determine the shapes of flexible blocks, and (6) to maximize yield. Since area minimization is a fundamental problem of VLSI design, in this paper, we restrict ourselves to the area minimization floorplan problem.

On the other hand, as the rapid increase of the transistor count in chip design, more and more design teams attempt to figure out an efficient way to combine their designs. Besides, designers in the same group are widely separated. The designers have to communicate each other by e-mail or by telephone. If there exists an efficient way to manage the designs with high fidelity, the designers can accelerate the design cycle. The Internet is a good choice. The salient feature of the Internet is distributed computing and resource sharing. Hence, a hierarchical design is especially suitable to be solved on the Internet.

Concerning these two issues together, in this paper, the authors try to solve the area minimization floorplan problem in the Internet environment.

We propose a novel algorithm in section 3, RMG algorithm, which takes advantage of the Internet to accelerate the computing of the floorplan for minimum area. RMG algorithm shortens the critical paths of computing and collaboratively completes the computing.

By solving this problem, it could be seen that the infrastructure of the Internet is suitable for EDA problems.

This paper is organized as follows. The preliminaries about floorplan are descirbed in Section 2. Section 3 formulates the problem as well as introduces RMG algorithm. The implementation and experimental results are shown in Section 4. Section 5 finally gives the conclusion of this paper.

2. PRELIMINARIES

2.1 Terminology

Before introducing the fundamental area minimization algorithm for floorplan, there exist six definitions as follows.

A **floorplan** for *n* modules is an enveloping rectangle which contains *n* non-overlapping rectilinear regions.

A **slicing floorplan** is a floorplan which is obtained by recursively subdividing a rectangle into two parts with either a horizontal or a vertical line.

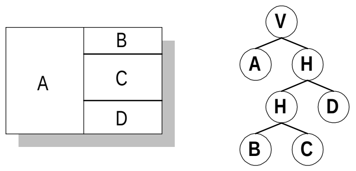

Figure 1: A slicing floorplan and the corresponding slicing tree

A **floorplan tree** is a binary tree in which an internal node represents a cut line and a leaf represents a module.

A **slicing tree** is a floorplan tree in which an internal node represents either a vertical or a horizontal line and a leaf represents a module. The corresponding floorplan of a slicing tree is a slicing floorplan depicted in Figure 1.

A **realization** for a module is a 4-tuple (*h1*, *h2*, *w1*, *w2*) in which *h1*, *h2*, *w1*, and *w2* represent the lengths of the left, right, top, and bottom edge respectively, as shown in Figure 2.

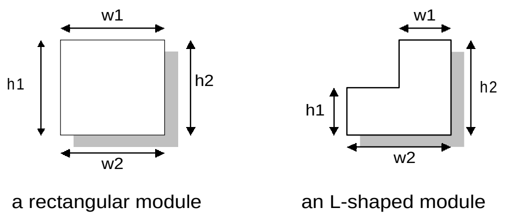

Figure 2: Realizations of two modules

An **irredundant realization list** of a module is a list of realizations, provided without two distinct realizations (*h1*, *h2*, *w1*, *w2*) and (*h1'*, *h2'*, *w1'*, *w2'*) such that

$$h1 \geq h1',\ h2 \geq h2',$$
$$w1 \geq w1',\ w2 \geq w2'.$$

Otten [1-2] pointed out a slicing floorplan is suitable for top-down hierarchical designs. Stockmeyer [3] showed the optimal algorithm for area minimization of a slicing tree. Specifically, the time complexity of this algorithm is O(*nd*), in which *n* is the number of modules and *d* is the depth of the slicing tree. In this paper, we adopt the representation of [4] which can represent rectangular and L-shaped modules. By this representation, slicing floorplans and wheels can be manipulated. Since this representation is in the form of a binary tree, it also has the hierarchy property of a slicing tree.

2.2 Node Types

From section 2.1, the internal nodes of a floorplan tree are corresponding to cut lines, while the leaves are corresponding to modules. The node type of a module can be represented by the N series nodes; however, the node type of a cut line can vary from V series, H series, M series, LV series and LH series as shown in Figure 3.

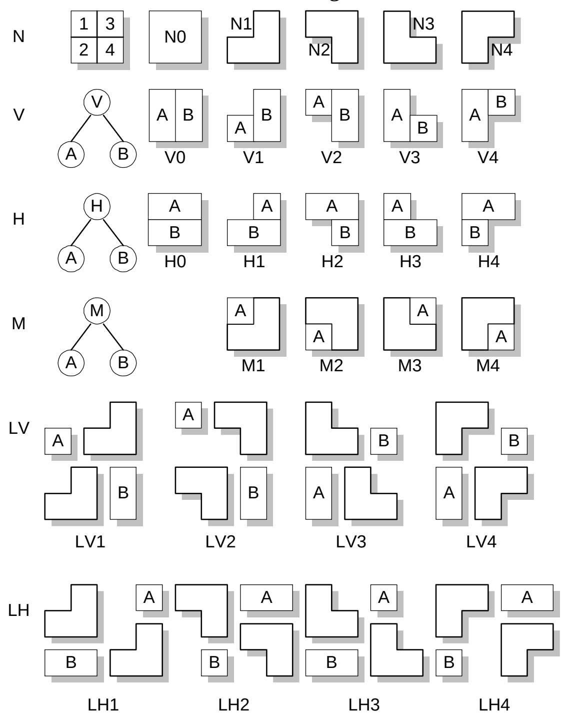

Figure 3: Node types

2.3 Hybrid Rectangular L-Shaped (HRL) Algorithm

After introducing the various node types, we present HRL algorithm [4] in this section. HRL algorithm expresses how to combine V, H, M, LV and LH series nodes. Each of them represents different combination method of two subtrees. HRL algorithm generates an optimal solution, the minimum area of an enveloping rectangle occupied by all modules. Here, we detail three typical combination algorithms.

V0 combination algorithm: Let n be a V0 node. According to the definition of V0 node, the left-child $n1$ and right-child $n2$ represent two N0 type nodes. The goal is to vertically combine $n1$ and $n2$. Let L_{n1} and L_{n2} be the sorted irredundant realization lists of $n1$ and $n2$. ($h1_{n1}(i)$, $h2_{n1}(i)$, $w1_{n1}(i)$, $w2_{n1}(i)$) expresses the i^{th} element of L_{n1}, and ($h1_{n2}(j)$, $h2_{n2}(j)$, $w1_{n2}(j)$, $w2_{n2}(j)$) expresses the j^{th} element of L_{n2}. The V0 combination algorithm constructs a new irredundant realization list L_n of n by sequentially combining L_{n1} and L_{n2} from the beginning to the end. The relationship between L_n, L_{n1} and L_{n2} is as follows.

$$h1_n(k) = h2_n(k) = \mathrm{Max}(h1_{n1}(i), h1_{n2}(j)),$$
$$w1_n(k) = w2_n(k) = w1_{n1}(i) + w1_{n2}(j),$$

where ($h1_n(k)$, $h2_n(k)$, $w1_n(k)$, $w2_n(k)$) is the k^{th} element of L_n.

Actually, the combination can be achieved with the time complexity O($n1$+$n2$). We don't have to consider all $n1 \times n2$ combinations because many of them are redundant.

V3 combination algorithm: Let n be a V3 node. By the definition of V3 node, the left child $n1$ and the right child $n2$ represent two N0 type nodes. The goal is to vertically combine $n1$ and $n2$ and forms an L-shaped (N3 type) module. Let L_{n1} and L_{n2} be the sorted irredundant realization lists of n_1 and n_2. ($h1_{n1}(i)$, $h2_{n1}(i)$, $w1_{n1}(i)$, $w2_{n1}(i)$) expresses the i^{th} element of L_{n1}, and ($h1_{n2}(j)$, $h2_{n2}(j)$, $w1_{n2}(j)$, $w2_{n2}(j)$) expresses the j^{th} element of L_{n2}. The V3 combination algorithm constructs a new irredundant realization list L_n by sequentially combining L_{n1} and L_{n2} from the beginning to the end as follows.

$$h1_n(k) = h1_{n1}(i),$$
$$h2_n(k) = h1_{n2}(j),$$
$$w1_n(k) = w1_{n1}(i),$$
$$w2_n(k) = w1_{n1}(i) + w1_{n2}(j),$$

where ($h1_n(k)$, $h2_n(k)$, $w1_n(k)$, $w2_n(k)$) is the k^{th} element of L_n.

A time complexity O($n1 \times n2$) is necessary for this algorithm. Note that the only exception is under the condition when Max($h1_{n1}(i)$, $h2_{n1}(i)$) < Max($h1_{n2}(j)$, $h2_{n2}(j)$).

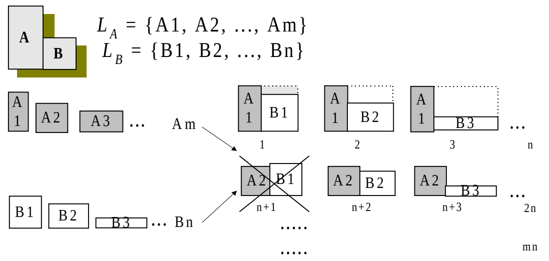

Figure 4: V3 combination algorithm

Figure 4 illustrates that the combination of modules A2 and B1 is unnecessary because it cannot generate an L-shaped module of N3 type as we expect.

Other V series and H series combination algorithms can be implemented in similar ways.

2.4 Transfer Latency

As we apply the floorplan algorithms on the Internet, some problems are introduced. The most obvious one might be the network transfer latency.

latency: For any internal node of a floorplan tree, the latency L of a node denotes time consumed to compute the optimal floorplan of the corresponding sub-tree rooted at this node. That is, $L = T + \mathrm{Max}(L_l, L_r)$, where T is the time consumed to perform combination, and L_l and L_r denote the latencies of the left and right children.

critical node: For any floorplan tree, a critical node is a leaf with maximum access time among all leaves.

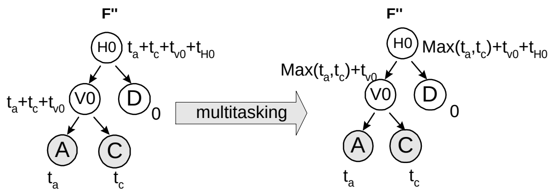

Figure 5: Timing improvement by multitasking

In order to speed up the computing of the minimum-area floorplan on the Internet, we introduce the multitasking technique widely used in the modern operation systems. In a multitasking system, users can simultaneously execute multiple programs. Taking advantage of multitasking, the total latency can be declined. For the example depicted in Figure 5, in the traditional algorithm without multitasking, the total latency is $t_a+t_c+t_{v0}+t_{H0}$. By using multitasking technique, because each node can fork two new processes to perform HRL algorithm simultaneously, the node V0 will have the latency of $Max(t_a,t_c)+t_{v0}$, and the total execution time of the floorplan is $Max(t_a,t_c)+t_{v0}+t_{H0}$.

3. PROBLEM AND RMG ALGORITHM

After introducing preliminaries, now we can formulate the problem as follows. Given a slicing and/or wheel floorplan, find the minimum area which covers the enveloping rectangle of all modules such that the computing time is minimum.

We introduce a novel algorithm, RMG algorithm, which stands for "Rotation, Modification and Grouping algorithm." In order to shorten the computing time, the objective is to raise the critical node in the floorplan tree to a level as high as possible without changing the floorplan. The three procedures of RMG algorithm are listed as follows.

- **Rotation algorithm** rotates the structure of given floorplan trees so that the critical node will be raised to a higher level in the floorplan tree.
- **Modification algorithm** modifies the node type of floorplan tree after rotation so that the original structure of the floorplan can be preserved.
- **Grouping algorithm** shrinks the whole subtree into a virtual node so that the floorplan tree can be simplified and easily analyzed with other nodes which have not been handled.

3.1 Rotation Algorithm

Rotation algorithm raises the critical node in a floorplan tree from a lower level to a higher one.

critical path: Let *F* be a floorplan tree. A critical path is a path from the root to the critical node.

L-L type floorplan tree: An L-L type floorplan tree *F* is a floorplan tree with the critical path {root→Left_child→Left_child}. The symbol "L" represents "Left," which expressess the direction of the critical path.

R-R, L-R and R-L type floorplan tree are defined in similar ways.

Figure 6 demonstrates the rotation algorithm. The rotation algorithm is very similar in spirit to the rotation of an AVL tree.

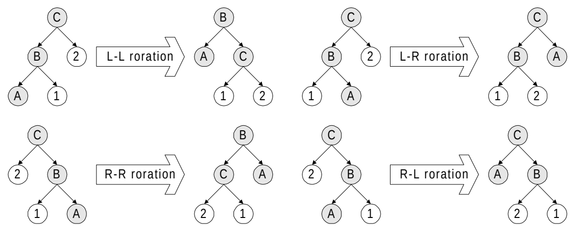

Figure 6: Rotation algorithm

3.2 Modification Algorithm

However, the floorplan structure may be inexact after rotation. Modification algorithm is to modify the node types of the rotated floorplan tree.

After modification, the corresponding floorplan of the modified floorplan tree occupies the same area with the original one. Besides, the modified subtree has a similar floorplan to the original subree. The floorplan of the modified subtree can be obtained by operating a mirror or a rotation on the corresponding subtree in the original floorplan. Therefore, we can recover the original floorplan according to the discrepancy.

Lemma 1: After applied rotation and modification algorithms to a subtree, the new floorplan of the subtree can be obtained by rotating or mirroring the original floorplan.

Figure 7 shows the example of modification algorithm; the new floorplan occupies the same area with the original one.

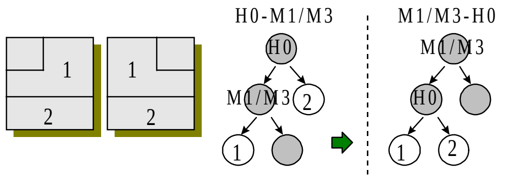

Figure 7: After rotation and modification, the resulting floorplan occupies the same area

3.3 Grouping Algorithm

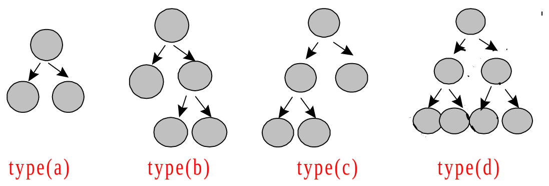

Figure 8: Four basic patterns of floorplan trees

Grouping algorithm is to group a subtree into a virtual node so that RMG algorithm will treat it as a leaf. The reason to virtually group a subtree is to simplify the floorplan tree so that RMG algorithm can perform optimization recursively. Another reason is that no improvement can be obtained by rotating the nodes inside this subtree.

The four basic patterns are illustrated in Figure 8. For type (a), no rotation has been performed so that there exists an opportunity to rotate it in the future. Hence, no grouping in this case.

For types (b) and (c), the critical node has been raised to a higher level after rotation algorithm, there exists no opportunity to raise other leaves. In this case, we can group this subtree into a virtual leaf.

Type (d) can be transformed to type (b) or (c) by grouping the left or the right subtree, depending on which of the access latency is larger. Applied rotation, modification and grouping recursively, each type of floorplan can be grouped into type (a) eventually.

3.4 An Overview of RMG Algorithm

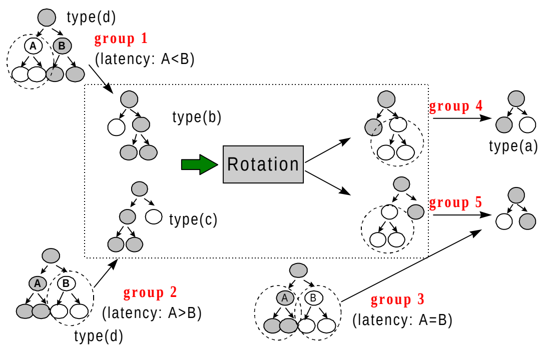

Figure 9: Overview of grouping algorithm

The RMG algorithm will recursively traverse from the root, and then the left subtree and the right subtree to find subtrees matching the basic patterns represented in Figure 8. If the patterns are found, RMG algorithm is performed to minimize the total execution time of subtrees. Figure 9 gives the global view of RMG algorithm; Figure 10 presents the pseudo code of RMG algorithm. As RMG algorithm collaborates with HRL algorithm, we can create an area-minimum floorplan within minimum execution time.

```
Algorithm: RMG algorithm
input: floorplan tree $T_i$;
output: floorplan tree $T_o$ with minimum total execution time;
begin
    switch(type of $T_i$)
    case 'leaf node','virtual node','type(a)': return;
    case 'type(b)','type(c)':
        rotation(root); modify(root);
        switch (type of root)
        case 'type(b)':grouping(root->right); break;
        case 'type(c)':grouping(root->left); break;
        root->latency=Max(root->left->latency,
                    root->right->latency); return;
    case 'type(d)':
        if (root->left->latency>root->right->latency) then
            grouping(root->right);
        else if (root->left->latency<root->right->latency)
            grouping(root->left);
        else
            grouping(root->right),grouping(root->left);
        root->latency=Max(root->left->latency,
                    root->right->latency); return;
    default:
        RMG(root->left),RMG(root->right);
end
```

Figure 10: RMG algorithm

4. EXPERIMENTAL RESULTS

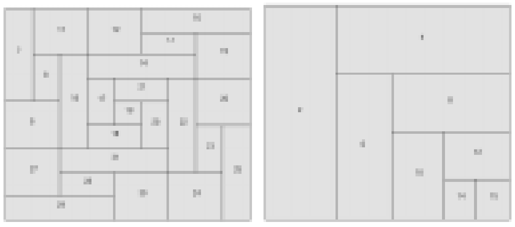
(a) (b)

Figure 11: (a)EX1-EX5 (b)EX6

We implemented RMG and HRL algorithms in C language on a Sun Sparc20 platform. RMG algorithm can solve the traditional floorplan problem by specifying all modules located in a local host. In other words, the program only performs HRL algorithm if restricted in a local host.

The floorplan with 25 rectangular modules shown in Figure 11(a) are used in the experiments from EX1 to EX5. The realization list of each module is shown as follows.

- EX1: {(4,4,1,1),(2,2,2,2),(1,1,4,4)}
- EX2: {(6,6,1,1),(3,3,2,2),(2,2,3,3),(1,1,6,6)}
- EX3: {(16,16,1,1),(8,8,2,2),(4,4,4,4),(2,2,8,8),(1,1,16,16)}
- EX4: {(12,12,1,1),(6,6,2,2),(4,4,3,3),(3,3,4,4),(2,2,6,6),(1,1,12,12)}
- EX5:{(24,24,1,1),(12,12,2,2),(8,8,3,3),(6,6,4,4),(4,4,6,6),(3,3,8,8),(2,2,12,12),(1,1,24,24)}

The execution time of HRL algorithm on each of the benchmarks is not only very small but also comparable to the traditional floorplan area minimization algorithms as captured in Table I.

The floorplan with 8 rectangular modules is shown in Figure 11(b). In experiment EX6, each module has eight possible realizations, {(24, 24, 1, 1), (12, 12, 2, 2), (8, 8, 3, 3), (6, 6, 4, 4), (4, 4, 6, 6), (3, 3, 8, 8), (2, 2, 12, 12), (1, 1, 4, 4)}.

Benchmarks	Total number of realizations	Execution time of HRL(sec)
EX1	3^{25}	2.4
EX2	4^{25}	2.6
EX3	5^{25}	2.9
EX4	6^{25}	3.3
EX5	8^{25}	5.1

Table I: Results of floorplan on local host

Since HRL can generate the floorplan with minimum area, the resulting area is not shown here. The experimental results in Table II show that the execution time can be reduced by 31.91% ~ 47.06%. The computation time is decrease by incorporating RMG algorithm into the HRL algorithm. Once the remote data access latency is longer than the computing time without latency, the execution time improved by RMG algorithm would be almost equal to the computing time without data access latency.

5. CONCLUSIONS

With the dramatic increase of transistor count in VLSI physical design, an efficient strategy to merge the design is needed. In this paper, we try to solve a well-known area-minimization floorplan problem on the Internet. The authors propose a novel algorithm, RMG algorithm. Taking advantage of distributed computing and resource sharing, RMG algorithm solves the area minimization of a hierarchical floorplan in the Internet environment. RMG algorithm reduces the execution time by shortening the critical path in the floorplan tree. With creating floorplan design in the Internet environment, designers can conveniently catch the latest design without wasting time in communication. By the example, it can be seen that the Internet favors Electronic Design Automation (EDA).

Bench-Marks	Condition	Execution Time(second)		
		(1) HRL algorithm	(2) HRL+RMG algorithm	(3) Improve-ment by (2) (1)-(2)/(1)
EX6	All nodes with latency 0	1.6	1.6	-
	Node 15 with latency 0.5	2.2	1.8	33.33%
	Node 15 with latency 1	2.7	1.8	40.74 %
	Node 15 with latency 1.6	3.4	1.8	47.06%
	Node 15 with latency 2	3.7	2.2	40.54%
	Node 15 with latency 3	4.7	3.2	31.91%

Table II: Results of floorplan in the Internet environment

6. REFERENCES

[1] Ralph H.J.M. Otten, "Automatic Floorplan Design," *19th Design Automation Conference*, pp. 261-267, 1982.

[2] Ralph H.J.M. Otten, "Layout Structures," *IEEE Large Scale Systems Symposium*, 1982.

[3] Larry Stockmeyer, "Optimal Orientations of Cells in Slicing Floorplan Designs," *Information and Control 57*, pp. 91-101, 1983.

[4] D. F. Wong and C. L. Liu, "Floorplan Design of VLSI Circuits," *Algorithmica*, Vol. 4, pp. 263-291, 1989.