MP-Trees: A Packing-Based Macro Placement Algorithm for Modern Mixed-Size Designs

Tung-Chieh Chen, *Student Member, IEEE*, Ping-Hung Yuh, Yao-Wen Chang, *Member, IEEE*, Fwu-Juh Huang, and Tien-Yueh Liu

***Abstract*—In this paper, we present a new multipacking-tree (MP-tree) representation for macro placements to handle modern mixed-size designs with large macros and high chip utilization rates. Based on binary trees, the MP-tree is very efficient, effective, and flexible for handling macro placements with various constraints. Given a global placement that already considers the areas and the interconnections among standard cells and macros, our MP-tree-based macro placer optimizes macro positions, minimizes the macro displacement from the initial macro positions, and maximizes the area of the chip center for standard-cell placement and routing. Experiments based on the Proceedings of the 2006 International Symposium on Physical Design placement contest benchmarks and Faraday benchmarks show that our macro placer combined with APlace 2.0, Capo 10.2, mPL6, or NTUplace3 for a standard-cell placement outperforms these state-of-the-art academic mixed-size placers alone by large margins in robustness and quality. In addition to wirelength, experiments on four real industrial designs with large macros and high utilization rates show that our method significantly reduces the average half-perimeter wirelength by 35%, the average routed wirelength by 55%, and the routing overflows by 13 times compared with Capo 10.2, implying that our macro placer leads to much higher routability.**

***Index Terms*—Floorplanning, layout, physical design, placement.**

I. Introduction

DUE TO the wide use of Intellectual Property modules and embedded memories, a modern very large-scale integration (VLSI) chip often consists of a significant number of large hard macros. The number of macros in a modern system-on-chip (SOC) design is dramatically increasing. Consequently, a modern SOC chip may consist of hundreds of hard macros, and a larger portion of the chip area, for example, more than 70%, may be occupied by hard macros [2]. Accordingly, mixed-size placements, which place both hard macros and standard cells, become more and more popular for real-world applications, and many mixed-size placement flows/algorithms are proposed in recent literature.

A. Previous Work

We can classify the mixed-size placement algorithms into three major types in terms of the macro handling methods. The first type simultaneously places macros and standard cells. A significant disadvantage of this type is that a robust macro legalizer is needed since macros are not guaranteed to be overlap-free after the placement stage. In particular, when the chip utilization rate is high, or some large macros exist, it is relatively much harder to legalize a placement. The simulated-annealing-based placers (Dragon [3], MBP [4], and mPG-MS [5]), the min-cut-based placers (Fengshui [6] and NTUplace [7]), and the analytical placers (APlace [8], Kraftwerk [9], mPL [10], and NTUplace3 [11], [12]) belong to this type.

The second type constructively places macros. Macros are guaranteed to be overlap-free during the placement process. Although this type of placers is usually more robust to find legal solutions, the wirelength (WL) is often much longer than those from the first type. The min-cut floorplacer, i.e., Capo [13], belongs to this type, for which fixed-outline floorplanning is applied when necessary during the min-cut placement process to ensure legal positions for macros.

The third type divides the mixed-size placement into two stages—a macro placement and then a standard-cell placement. The macro positions are first determined, and then standard cells are placed into the remaining area. We summarize prior *macro placement* approaches and their characteristics in Table I. Two combinatorial techniques were proposed in [14]. The first approach uses a standard-cell placer to obtain an initial placement. Standard cells are clustered as soft macros based on the initial placement, and fixed-outline floorplanning is applied to find an overlap-free macro placement. The second approach uses a standard-cell placer to obtain an initial placement and a force-directed method to remove macro overlaps. Recently, Auletta [15] has developed an edge placement expert system for macro placements/floorplanning. This method requires many predefined rules for a macro placement and does not provide a systematic approach to optimize the

Manuscript received September 2, 2007; revised January 6, 2008 and March 23, 2008. Published August 20, 2008 (projected). This work was supported in part by Etron, MediaTek, Realtek, SpringSoft, TSMC, UMC, and the National Science Council of Taiwan under Grants NSC 96-2628-E-002-248-MY3, NSC 96-2628-E-002-249-MY3, NSC 96-2221-E-002-245, and NSC 96-2752-E-002-008-PAE. This paper was presented in part at the 2007 Association for Computing Machinery/IEEE Design Automation Conference, June 2007. This paper was recommended by Associate Editor H. Onodera.

T.-C. Chen is with the Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 106, Taiwan, R.O.C. (e-mail: donnie@eda.ee.ntu.edu.tw).

P.-H. Yuh is with the Department of Computer Science and Information Engineering, National Taiwan University, Taipei 106, Taiwan, R.O.C. (e-mail: r91089@csie.ntu.edu.tw).

Y.-W. Chang is with the Department of Electrical Engineering, National Taiwan University, Taipei 106, Taiwan, R.O.C., and also with the Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 106, Taiwan, R.O.C. (e-mail: ywchang@cc.ee.ntu.edu.tw).

F.-J. Huang and T.-Y. Liu are with MediaTek Inc., Hsinchu 300, Taiwan, R.O.C. (e-mail: fj_huang@mtk.com.tw; denny_liu@mtk.com.tw).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2008.927760

TABLE I
MACRO PLACEMENT COMPARISONS

Approach	Characteristics
Adya and Markov Flow 1 [14]	• Use fixed-outline floorplanning to find macro positions. • Is robust in finding overlap-free placements.
Adya and Markov Flow 2 [14]	• Use the force-directed method to remove macro overlaps. • May result in overlaps in final placements.
Auletta [15]	• Use pre-defined rules to determine macro positions. • Place macros along the boundary of the chip. • Lack a systematic approach to optimize placement.
Our MP-tree	• Use a packing-based method to optimize macro positions. • Place macros along the boundary of the chip. • Is robust in finding overlap-free placements.

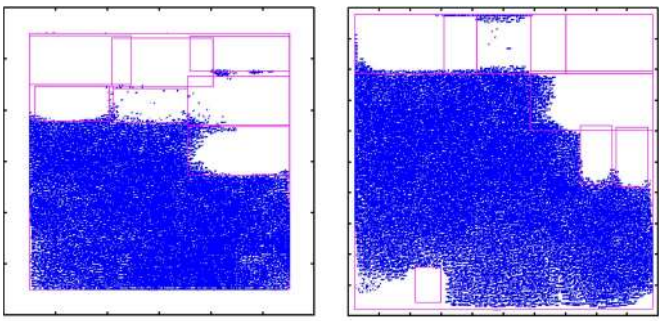


Fig. 1. Global placement result generated by APlace. Since macros are not handled well, it is not easy to remove those overlaps without some fundamental changes in the placement. (a) RISC1. (b) RISC2.

placement. With the increasing design complexity in modern SOCs, a systematic approach for macro placements is particularly desired.

Compared with the first two types of mixed-size placement approaches, the two-stage mixed-size placement is more robust since it can guarantee a feasible solution as long as an overlap-free macro placement is obtained. In particular, the recent trends of a placement with large macros and high utilization rates make the placement problem even harder. The sizes of these macros are usually more than 1000 times larger than those of standard cells, and the chip utilization rate is getting higher to reduce the die area to decrease the cost. A poor macro placement may not only cause large overlaps but also degrade the WL and/or the performance. Fig. 1 shows the global placements generated by APlace for the circuits RISC1 and RISC2 of the Faraday benchmark suite [16], [17]. As we can see from the layouts, when macros are not placed well, it is not easy to remove these overlaps without some substantial changes in the placement. As a result, the two-stage approach is widely used in the industry. Because of the advantages of the third-type placers and its systematic mechanism, we adopt the two-stage mixed-size placement approach. This paper focuses on the first stage, i.e., the macro placement, which is crucial for mixed-size placements since macro positions significantly affect the placement of standard cells and the final placement and routing quality.

B. Our Contributions

In this paper, we present a new *multipacking-tree* (*MP-tree*) representation for macro placements in mixed-size designs, particularly with large macros and high chip utilization rates. Given a global placement that already considers the areas and the interconnections among standard cells and macros, our MP-tree-based macro placer optimizes macro positions to remove overlaps, minimizes the macro displacement from the initial macro positions, and maximizes the area of the chip center for a standard-cell placement and routing. Our studies show that the publicly available state-of-the-art placers alone often cannot handle this kind of "difficult'" mixed-size designs well. In contrast, we can combine our MP-tree and a state-of-the-art placer to form a comprehensive placer for mixed-size designs. In other words, the MP-tree and these placers can be complementary to each other for modern mixed-size designs. We summarize the advantages of the MP-tree-based macro placement algorithm as follows.

1) Our macro placement is based on a packing technique, so that we can easily find legal solutions even for the cases with large macros, a large number of macros, and/or high chip utilization rates. Also, the MP-tree can easily handle various placement constraints commonly seen in real industry designs, such as preplaced blocks, corner blocks, placement blockages, and region constraints. Thus, our macro placer is very flexible for practical applications.
2) Based on binary trees, the MP-tree is very fast for operations and packing. It only needs amortized linear time to transform an MP-tree to its corresponding macro placement result. Thus, we can additionally consider many design constraints and efficiently search the solutions by simulated annealing.
3) The MP-tree structure directly induces a special hierarchical framework for the optimization of macro placement. Each MP-tree can be subdivided into a set of packing subtrees, with each subtree handling macro packing to a corner (*local optimization*). A major drawback of the traditional hierarchical framework is that it lacks the global information for the interaction among subtrees (subproblems). Because of the branch structure in the MP-tree, unlike the traditional hierarchical framework, the interaction between different subtrees of an MP-tree is well preserved, facilitating the *global optimization* among all subtrees. Experiments show that the MP-tree obtains an 8% shorter average WL than that obtained by the traditional hierarchical method using four independent packing trees.
4) Our two-stage mixed-size placement methodology is robust for finding a desirable result. By minimizing the macro displacement during the macro placement process, we can keep a smaller WL. By reserving the chip center for a standard-cell placement and routing, our macro placement results in better routability and a shorter routed WL.
5) The MP-tree combined with leading academic placers can robustly generate feasible results even for the designs with 95% chip utilization, whereas the leading mixed-size

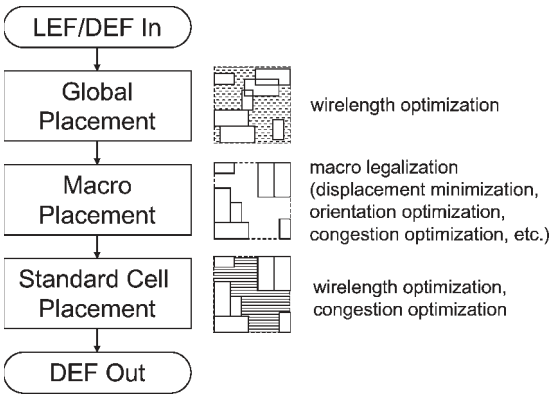


Fig. 2. Our mixed-size placement flow.

placers alone sometimes fail to find a legal result. Furthermore, the MP-tree with the placers results in a significantly shorter WL. Based on the eight Proceedings of the 2006 International Symposium on Physical Design (ISPD '06) placement contest benchmarks [18] and 90% chip utilization, the MP-tree combined with Capo 10.2 can reduce the half-perimeter WL (HPWL) by 12% than Capo alone, resulting in an HPWL of 12%, on average; the MP-tree combined with NTUplace3 can fix two illegal placements and further reduce the HPWL by 7%, on average, for the other six legal placements, and the MP-tree combined with mPL6 can fix seven illegal placements and further reduce the HPWL by 4% for the only legal placement. Similar improvements in the legality and the WL can be found based on the four Faraday benchmarks [16] with large macros and high utilization rates.

6) In addition to the placement, we also tested four real industrial designs up to the routing stage. Based on the same standard-cell placer, our MP-tree-based macro placer can significantly reduce the average HPWL by 35%, the average routed WL by 55%, and the routing overflows by 13 times compared with Capo 10.2. This result implies that our macro placer can lead to much higher routability.

The rest of this paper is organized as follows. Section II presents our floorplan representation for the macro placement. Section III presents our macro placement algorithm based on the MP-tree. Section IV provides the solution space and the reachability of the MP-tree. Section VI presents the method of handling several placement constraints. The experimental results are given in Section V. Last, the conclusion is given in Section VI.

II. Macro Placement

We adopt the two-stage mixed-size placement flow: 1) a macro placement followed by 2) a standard-cell placement. See Fig. 2 for our mixed-size placement flow. After the circuit information is imported, a WL-driven global placement is applied to find global macro positions. Based on the given macro positions, our macro placer then determines the legal positions of macros and places macros along the chip boundary. With the objective of the macro displacement minimization, our macro placement algorithm results in better macro positions that lead to a smaller WL increase for the later stage implicitly. The macro placement step plays an important role in the mixed-size placement. Last, all macros are fixed, and a standard-cell placer is then applied to place all standard cells in the available space.

Although our flow is the same as that of Adya and Markov [14], the two approaches are very different. We summarize two major differences as follows.

1) We do not shred macros into standard cells in the global placement stage, whereas Adya and Markov [14] do. Shredding macros incurs some negative effects, such as additional constraints in the later stage to enforce that all standard cells associated with a macro must be placed together, and also inaccurate estimation of the WL between macros and standard cells. This inaccuracy is more severe if the routing resources associated with macros need to be considered; macros typically incur blockages that block the interconnects running above the macros. Consequently, it is much harder to accurately estimate the routing resources with macro shredding.
2) In [14], it handles not only macros but also standard cells at the floorplanning stage. The standard cells are clustered as soft macros. However, it tends to increase the problem size, and, thus, it is harder to apply simulated annealing to find a desired placement. In addition, their objective function considers only the WL, ignoring the routing blockages induced by macros. As a result, their generated floorplans may lead to lower routability. In contrast, we place macros on the chip boundary and reserve a continuous region without routing blockages for a standard-cell placement. Therefore, our method often leads to higher routability.

For the two-stage approach, placing macros along the boundary of a chip or resorting to a hierarchical partition is a common and popular practice for both flat application-specific integrated circuits (ASICs) and hierarchical SOCs. This approach can create a regular region for the standard-cell placement. Furthermore, there are routing blockages above macros in real-world applications, and the macros tend to block the routes if they are placed in the chip center. Also, by minimizing the macro displacement, we can implicitly minimize the increased WL since the given global placement has been optimized for the WL.

The traditional packing floorplanning techniques cannot be directly applied to the macro placement problem since they usually pack all macros to one corner. To overcome this problem, we propose a new MP-tree floorplan representation to place macros along the given region boundary. We shall first present the packing-tree floorplan representation.

A. Packing-Tree Floorplan Representation

A packing tree is a binary tree for modeling nonslicing or slicing floorplans. Each node in the packing tree corresponds to a macro. There are four types of packing for a packing tree. BL-, TL-, TR-, and BR-packing subtrees pack the blocks to the bottom-left, top-left, top-right, and bottom-right corners, respectively.

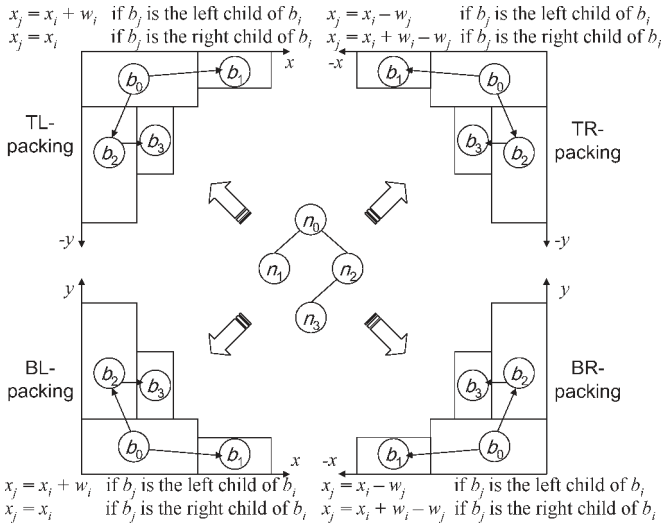


Fig. 3. Packing tree with its four types of packing.

Fig. 3 shows a packing tree and its corresponding four packing types. Let $(x_{\text{corner}}, y_{\text{corner}})$ be the coordinate of a corner (there are four corners in a rectangular region), (x_i, y_i) be the bottom-left coordinate of the block b_i, and w_i (h_i) be the width (height) of the block b_i. The coordinate of the root of a packing tree is at:

1) $(x_{\text{corner}}, y_{\text{corner}})$ for BL-packing;
2) $(x_{\text{corner}}, y_{\text{corner}} - h_{\text{root}})$ for TL-packing;
3) $(x_{\text{corner}} - w_{\text{root}}, y_{\text{corner}} - h_{\text{root}})$ for TR-packing;
4) $(x_{\text{corner}} - w_{\text{root}}, y_{\text{corner}})$ for BR-packing.

Given a packing tree, the x-coordinates of all blocks can be determined by traversing the tree in linear time. The x-coordinate of node b_j can be computed from its parent node b_i according to the type of the packing tree. If node n_j is the right child of n_i, the block b_j is:

1) the lowest adjacent block on the right with $x_j = x_i + w_i$ for BL-packing;
2) the highest adjacent block on the right with $x_j = x_i + w_i$ for TL-packing;
3) the highest adjacent block on the left with $x_j = x_i - w_j$ for TR-packing;
4) the lowest adjacent block on the left with $x_j = x_i - w_j$ for BR-packing.

If node n_j is the left child of n_i, the block b_j is:

1) the first block above b_i with $x_j = x_i$ for BL-packing;
2) the first block below b_i with $x_j = x_i$ for TL-packing;
3) the first block below b_i with $x_j = x_i + w_i - w_j$ for TR-packing;
4) the first block above b_i with $x_j = x_i + w_i - w_j$ for BR-packing.

Furthermore, a y-coordinate can be computed using the contour data structure in amortized constant time, similar to the method used in [19]. Therefore, the complexity of transforming a packing tree to the corresponding placement is amortized linear time. Note that B*-tree floorplan representation [19] is a BL-type packing tree.

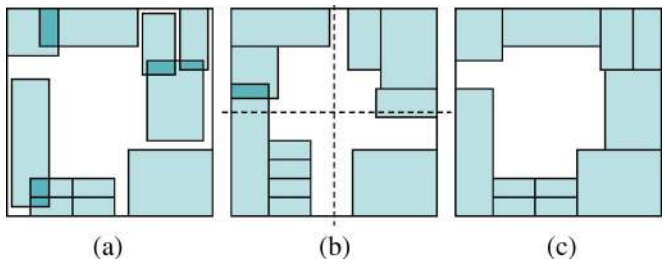


Fig. 4. (a) Initial macro placement. (b) Macro placement by dividing the chip into four regions. The macros in the four regions are independently placed, and, thus, it lacks the global view of the placement interactions among different regions. (c) Better macro placement without dividing the chip into subregions.

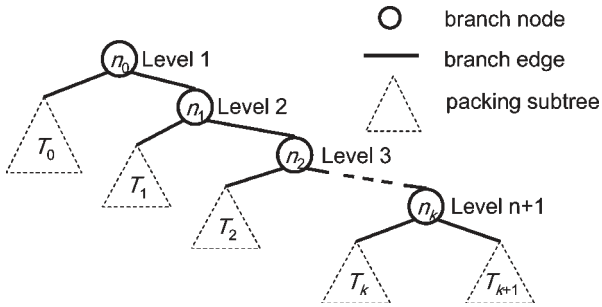


Fig. 5. General MP-tree.

A packing tree handles only one direction of packing, and, thus, it is not suitable for our macro placement since all macros would be packed toward a chip corner. In the following section, we extend the packing tree to handle a macro placement with placement constraints.

B. MP-Tree Floorplan Representation

Since a packing tree always packs macros to a corner, we could apply the traditional hierarchical method by subdividing the chip into four regions and create four packing trees to handle different regions. However, this approach has some limitations. First, the macros in a region must be placed inside the given region, which is overconstrained because there is no real boundary between the regions. Furthermore, assigning the regions for macros greatly affects the resulting placement because a macro cannot change its region once its region is assigned. As a result, we may not obtain a desirable placement because there is no interaction among different regions.

Fig. 4 illustrates the disadvantages of solving the macro placement problem by subdividing the chip. Given an initial macro placement in Fig. 4(a), Fig. 4(b) shows a possible floorplan result by dividing the chip into four subregions and performing floorplanning in each region based on the packing-tree representation. To satisfy the fixed-outline constraint, all macros must be placed inside their regions, which may incur a large macro displacement. Furthermore, there may be some large macros that can never fit into their regions, thus causing significant macro overlaps. Instead, a better alternative is to optimize all macros at the same time and globally consider the interactions among regions. Fig. 4(c) gives a better solution with a smaller macro displacement by employing the global optimization technique.

```
01. function ComputeX(node n);
02.     if(n == rootOfPackingSubtree)
03.         if(n.corner.type == BL) n.x = n.corner.x;
04.         else if(n.corner.type == BR) n.x = n.corner.x − n.w;
05.         else if(n.corner.type == TR) n.x = n.corner.x − n.w;
06.         else n.x = n.corner.x; // TL
07.     else if(n.type != branchNode)
08.         if(n == n.parent.leftChild)
09.             if(n.corner.type == BL) n.x = n.parent.x + n.parent.w;
10.             else if(n.corner.type == BR) n.x = n.parent.x − n.w;
11.             else if(n.corner.type == TR) n.x = n.parent.x − n.w;
12.             else n.x = n.parent.x + n.parent.w; // TL
13.         else // n == n.parent.rightChild)
14.             if(n.corner.type == BL) n.x = n.parent.x
15.             else if(n.corner.type == BR) n.x = n.parent.x
16.             else if(n.corner.type == TR)
17.                 n.x = n.parent.x + n.parent.w − n.w
18.             else n.x = n.parent.x + n.parent.w − n.w // TL
19.     if(n.leftChild != NULL) ComputeX(n.leftChild);
20.     if(n.rightChild != NULL) ComputeX(n.rightChild);
```

Fig. 6. Recursive function to compute the x-coordinates.

```
01. function ComputeY(node n);
02.     if(n.type != branchNode)
03.         if(n.corner.type == BL or BR)
04.             n.y = GetMaxYBelowMacroInBottomContour(n);
05.             UpdateBottomContour(n);
06.         else // n.corner.type == TL or TR)
07.             n.y = GetMaxYAboveMacroInTopContour(n);
08.             UpdateTopContour(n);
09.     if(n.leftChild != NULL) ComputeY(n.leftChild);
10.     if(n.rightChild != NULL) ComputeY(n.rightChild);
```

Fig. 7. Recursive function to compute the y-coordinates.

To implement the global optimization idea, we resort to the MP-tree to handle the global interaction among different regions. An MP-tree combines several packing trees for different corners. We use *branch nodes* to connect different packing subtrees.

Fig. 5 shows an example of a general MP-tree. There are k branch nodes in an MP-tree to integrate $k+1$ packing subtrees. We use a right-skewed branch to integrate the packing subtrees for the purpose of easier implementation. By doing so, the packing order of the subtrees can be determined by the level of the parent node of the packing subtrees. With the depth-first search (DFS) order of the tree traversal for packing, the smaller the level is, the earlier the packing subtree packs blocks. If the parents of two packing subtrees are the same, the left packing subtree will be handled first. The general MP-tree can be used to model the placement in any rectilinear floorplan region, with each packing subtree packing to one convex corner.

The MP-tree structure directly induces a special hierarchical framework for the optimization of the macro placement. Each packing subtree handles macros being packed to a corner, i.e., performs *local optimization*. A major drawback of the traditional hierarchical framework is that it lacks the global information for the interaction among subtrees (subproblems). Because of the branch structure in the MP-tree, unlike the traditional hierarchical framework, the interaction between different subtrees of an MP-tree is well preserved, facilitating the *global optimization* among all subtrees. It will be clear in Section V that the MP-tree leads to significantly better placement quality

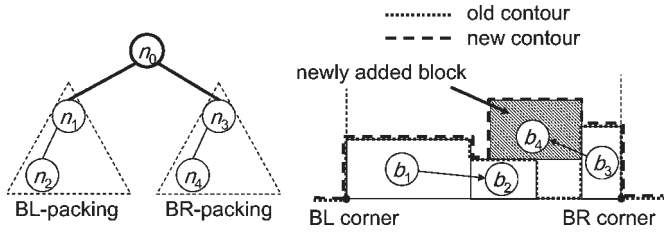


Fig. 8. Packing example for an MP-tree with a BL-packing subtree and a BR-packing subtree. Adding a new block b_4 to the placement, we search the contour and update it with the top boundary of the new block.

than that obtained by the traditional hierarchical method using independent packing trees.

To transform an MP-tree to its corresponding placement, the coordinates of blocks can be determined by a DFS traversal. Fig. 6 gives a recursive algorithm to compute the x-coordinates from the root of a given MP-tree. If the given node is the root of a packing subtree, we use the corner coordinate and the tree type to determine the x-coordinate of the block (lines 2–6). If the given node is not a branch node, we can determine the x-coordinate of the block based on its relation with the parent node (lines 7–18). After determining the x-coordinate of a node, we continue the traversal from its left child and then its right child (lines 19 and 20).

Fig. 7 gives an algorithm for computing the y-coordinates. To compute the y-coordinates, we keep two contours—the bottom contour and the top contour—which are initialized according to the bottom side and the top side of the given floorplan

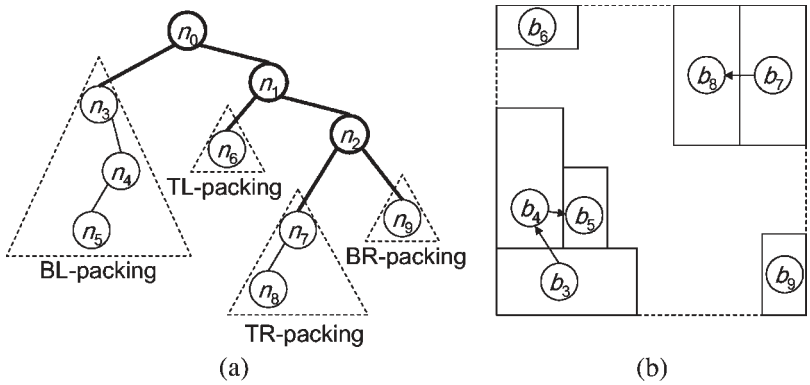


Fig. 9. (a) MP-tree with four packing subtrees. (b) Corresponding macro placement.

region, respectively. Both BL- and BR-packing subtrees use the bottom-contour data structure (lines 3–5), whereas the TL- and TR-packing subtrees use the top-contour one (lines 6–8). The packing subtrees that use the same contour data structure always generate overlap-free placement results since the contour reserves the spaces of the traversed blocks. BL-/BR-packing subtrees, however, may overlap with TL-/TR-packing subtrees, and, thus, we should discard this kind of an infeasible solution. The y-coordinate is also determined in a recursive manner. After determining the y-coordinate of a node, we continue the traversal from its left child and then its right child (lines 9 and 10).

Fig. 8 gives a packing example for an MP-tree with BL- and BR-packing subtrees. The packing order is b_1, b_2, b_3, and b_4. Adding a new block b_4 to the placement, we search the contour and update it with the top boundary of the new block. It is clear that no overlap will occur when we process the BL- and BR-packing subtrees.

For a rectangular VLSI chip, we can use an MP-tree with four packing subtrees to handle it, as shown in the example in Fig. 9(a). To obtain the corresponding macro placement, we traverse the tree in the DFS order from the root n_0. Since n_0 is a branch node, we do nothing and continue the traversal. Then, the left child of n_0, i.e., n_3, is the root of the BL-packing subtree; therefore, we place b_3 on the bottom-left corner. Since n_3 does not have a left child, we traverse n_4 and continue the traversal. In this example, the packing subtrees are traversed in the order of the BL-packing subtree, the TL-packing subtree, the TR-packing subtree, and, finally, the BR-packing subtree. After we traverse all nodes, the macro placement shown in Fig. 9(b) is obtained.

III. Macro Placement Algorithm

A. Macro Placement Flow

Fig. 10 shows the flow for our MP-tree macro placer. This flow readily extends to the macro placement with various constraints, such as rectilinear macros, preplaced macros, placement blockages, and macro clustering. For easier presentation, however, we shall focus on the macro placement in this section and discuss these constraints in the Appendix.

After reading library exchange format/design exchange format files, we cluster the macros under the designated *performance constraints*, and the cluster dimension is initialized with

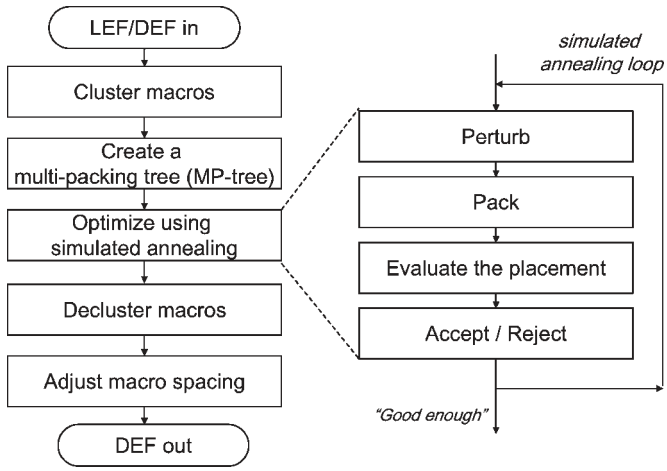


Fig. 10. Our MP-tree macro placement flow. Note that this flow is the second step, i.e., the macro placement, of the flow of Fig. 2.

the one closest to square since the square shape usually leads to better results. Then, we create an MP-tree with its number of packing subtrees equal to the number of the corners in the placement region. If a *region constraint* is given, we need to create four subtrees for the region. Each macro/cluster corresponds to a node in a packing subtree. We assign an initial packing subtree to a tree node corresponding to the nearest corner to which the macros are placed in the given global placement. Each packing subtree is then initialized as a complete binary tree.

Simulated annealing is then used to find a desired macro placement. We perturb one MP-tree to another by the operations described in the next section. After the perturbation, we fix the tree structure to satisfy the given macro placement constraints, pack the MP-tree, evaluate the macro placement, and decide whether we should accept the new solution according to the difference of the macro placement quality and the current temperature of simulated annealing. Then, the MP-tree is perturbed again. The simulated annealing continues until the solution is good enough or no better solution can be found, and all positions of blocks/clusters are determined. Then, the positions of macros inside a cluster can be computed according to the matrix dimension of the cluster.

To reserve enough spacing between macros, we use the number of pins along the macro boundary to estimate the routing resource. Before the macro placement, macros are

enlarged to reserve the spacing. The spacing is computed by the product of the number of pins and the wire spacing divided by available routing layers. A channel routing algorithm can also be applied to more accurately estimate the routing resource. If the demand of the routing resource between two macros is higher than the original spacing, we add more space between these two macros; otherwise, we can reduce the original spacing to obtain a more compact placement. After floorplanning, we further use a heuristic to flip macros. The basic idea is that the majority of interconnections of a macro are among standard cells and this macro for practical applications. Therefore, we vertically/horizontally flip a macro to find an orientation to make the majority of its pin face the center of the chip. This way, we can effectively minimize the WL among standard cells and macros, and, thus, the total WL. Last, we fix all macros and report the final macro placement solution.

B. Operations on MP-Tree

We define the perturbation operations of the MP-tree for use in simulated annealing. An MP-tree is perturbed to get another MP-tree by the following operations.

1) Op1: rotate a block or a cluster.
2) Op2: resize a cluster.
3) Op3: move a node in a packing subtree to another place.
4) Op4: swap two nodes within one or two packing subtrees.
5) Op5: swap two packing subtrees.

For Op1, we rotate a block or a cluster for a tree node. For Op2, we change the clustering dimension of a cluster. Note that Op1 and Op2 do not affect the MP-tree structure; instead, they only change the information within the node. Therefore, their time complexity is $O(1)$. For Op3, we select a node from a packing subtree and move it to another place of the same or different packing subtrees. Two steps are needed for the move—deletion and insertion. The time complexity of the deletion is $O(h)$, where h is the height of the node being deleted, whereas the time complexity of the insertion is $O(1)$. For Op4, we select two nodes from one (two) packing subtree(s) and swap them. For Op5, we swap two packing subtrees and exchange the packing order of the two packing subtrees. For this perturbation, we only need to exchange the pointers of the nodes for Op4 and Op5, and, thus, the time complexity is $O(1)$. Note that the branch structure of the MP-tree is not changed by any type of operations.

C. Evaluation of a Macro Placement

To evaluate the quality of a macro placement solution, the cost of a macro placement F is defined as follows:

$$\Phi(F) = \alpha A + \beta W + \gamma D + \delta O \tag{1}$$

where A is the *macro placement area*, W is the total WL, D is the total *macro displacement*, O is the *vertical overlap length*, and α, β, γ, and δ are user-specified weighting parameters. The macro placement area, the WL, the macro displacement, and the vertical overlap length are explained in the following.

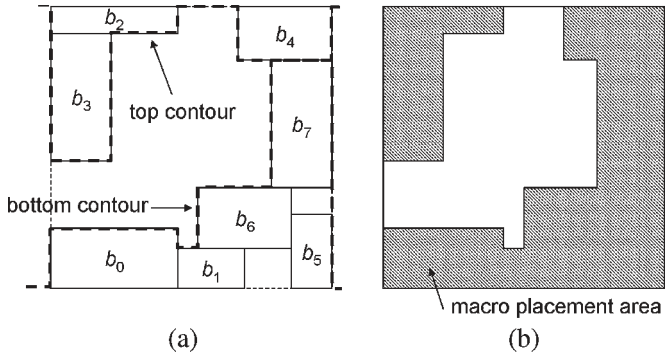


Fig. 11. (a) Macro placement solution and its top and bottom contours. (b) Corresponding macro placement area.

The macro placement area is defined as the area under the bottom contour plus the area above the top contour. As shown in Fig. 11(a), the contours are plotted by dashed lines, and the corresponding macro placement area is shown in Fig. 11(b). Minimizing the macro placement area can make more space for the standard cells in the central region of the chip. By doing so, the routing between standard cells will be easier, and, thus, the routed WL will be smaller.

For the WL, since we consider only macros during the placement, we shall not directly use the netlist from the circuit. Instead, we create pseudonets among macros of the same design hierarchy group based on the star or the clique models [20]. Therefore, minimizing the WL of these pseudonets can pack the macros of the same design hierarchy closer.

The macro placement should honor the given global placement since the global placement is optimized for the WL and other objectives. We extract the given macro positions from the global placement and use the macro displacement as a penalty of the cost function. By doing so, we can find a desired macro placement with the minimum macro displacement. The cost D is the total macro displacement, which is defined by

$$D = \sum_{\text{blocks}} \left(|x_i' - x_i| + |y_i' - y_i|\right)^2 \tag{2}$$

where (x_i, y_i) is the given position of the macro b_i, and (x_i', y_i') is the current position of the macro b_i during the simulated annealing. The quadratic penalty can prevent a single macro from having a large displacement.

Our MP-tree representation can guarantee no overlaps between the top and bottom packing subtrees. However, there may exist vertical overlaps between the top and bottom contours. The penalty cost O for the vertical overlap can guide the simulated annealing to find a nonoverlap solution.

IV. Solution Space and Reachability

A. Solution Space

We have the following theorem for the MP-tree solution space.

Theorem 1: The size of the solution space for an MP-tree is $O(mn!2^{2n}/n^{1.5})$, where m is the number of packing subtrees, and n is the number of nonbranch nodes.

Proof: The total number of combinations of an MP-tree can be computed by the number of unlabeled binary trees and the permutation of n labels. Suppose that we have n macros to be packed to m corners. As a result, there are m subtrees in the MP-tree, and each subtree has n nodes at most. The permutation of n labels is $n!$. From [21], the counting of an unlabeled p-ary tree with n node is

$$\frac{1}{(p-1)n+1}\binom{pn}{n}. \tag{3}$$

Applying Stirling's approximation, we have

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n. \tag{4}$$

Setting p to 2 in (3), we can obtain the following asymptotic form:

$$O\left(\frac{2^{2n}}{n^{1.5}}\right). \tag{5}$$

Thus, the total number of possible placements for an MP-tree with m subtrees and n nonbranch nodes is

$$O\left(mn!\frac{2^{2n}}{n^{1.5}}\right). \tag{6}$$

■

B. Reachability

A well-structured solution space should have the property that there exist a series of operations to transform from one solution to another. For such a solution structure, it is possible to find an optimal solution from any initial solution in the solution space. Two macro placements are *equivalent* if the topologies of their corresponding MP-trees, the label of each node, and the orientations of all blocks are the same.

Theorem 2: Given two MP-trees M_1 and M_2, M_1 can be transformed to M_2 via at most $3n$ operations.

Proof: We use the DFS traversal order to transform M_1 to M_2. Each time, we check the nodes in M_1 and M_2 at the same position. If two nodes are the same, we do not need to do any operation and continue the traversal. If the node in M_1 is different from that in M_2, we find the correct node in M_1 and swap them (Op4). If there is no node in M_1 at the corresponding position of M_2, we find the correct node in M_1 and move it to the correct position (Op3). After the DFS traversal, the structure and the labels of M_1 are the same as those of M_2. Thus, transforming the structure and the labels requires at most n operations. If clusters exist, we can change the cluster dimension (Op2) with at most n operations. The final step is to rotate blocks or clusters, which also needs at most n operations. Therefore, the total number of the operations, including changing the tree structure, labels, cluster dimensions, and orientations, is $3n$ at most. ■

C. Comparison With Other Floorplan Representations

Tables II and III give the solution space and the worst case number of operations to transform one floorplan instance to another for MP-tree, B*-tree [19], Q-sequence [22], slicing tree (normalized Polish expression) [23], Sequence Pair [24], TCG [25], and TCG-S [26]. Note that our MP-tree representation can pack macros to several corners, which is different from all other representations that pack macros to only one corner. From these two tables, we can see that the MP-tree has a relatively smaller solution space and a shorter worst case distance (in terms of the number of operations required) to transform one floorplan instance to another. Therefore, the MP-tree does have significant advantages to be applied with a nondeterministic algorithm, such as simulated annealing.

TABLE II
SOLUTION SPACE COMPARISON FOR n MACROS. m IS THE NUMBER OF CORNERS FOR MP-TREE PACKING. OTHER REPRESENTATIONS CAN ONLY PACK MACROS TO ONE CORNER

Representation	Solution Space
MP-tree	$O(mn!2^{2n}/n^{1.5})$
B*-tree	$O(n!2^{2n}/n^{1.5})$
Q-sequence	$O(2^{6n}(2n)!/n!)$
Slicing-tree (NPE)	$O(n!2^{3n}/n^{1.5})$
Sequence Pair (SP)	$(n!)^2$
TCG	$(n!)^2$
TCG-S	$(n!)^2$

TABLE III
REACHABILITY COMPARISON FOR n MACROS

Representation	#Operations for transformation
MP-tree	$O(n)$
B*-tree	$O(n)$
Q-sequence	$O(n)$
Slicing-tree (NPE)	$O(n)$
Sequence Pair (SP)	$O(n)$
TCG	$O(n^2)$
TCG-S	$O(n^2)$

V. EXPERIMENTAL RESULTS

To show the effectiveness and the robustness of the MP-tree, we conducted five experiments on an Opteron 2.6-GHz machine based on three sets of benchmark circuits with *large macros* and various chip utilization rates. We used several state-of-the-art publicly available academic mixed-size placers, including APlace 2.0 [27], Capo 10.2 [13], Fengshui 5.1 [28], mPL6 [10], and NTUplace3 [11], [12]. According to the macro handling method in Section I-A, APlace, Fengshui, mPL, and NTUplace3 belong to the first type, and Capo belongs to the second type. We also integrated our MP-tree with those mixed-size placers to study the effectiveness of our macro placer.

For the first experiment, we studied the effects of different chip utilizations on the MP-tree by combining the MP-tree with NTUplace3. In the second experiment, we compared our MP-tree with the traditional hierarchical method using four independent packing trees. In the third experiment, we further combined the MP-tree with mPL and Capo to evaluate the effectiveness of the MP-tree with different placers using the ISPD '06 benchmarks [18]. In the last two experiments, we adopted the Faraday benchmarks [16], [17] and four real industry designs to evaluate the WL and the routability of various placers up to the routing stage.

TABLE IV
STATISTICS OF THE ISPD '06 BENCHMARKS

Circuit	# Cells	# Nets	# Macros	MA-ratio
adaptec5	842k	868k	58	53%
newblue1	330k	339k	21	49%
newblue2	436k	465k	9	57%
newblue3	482k	552k	18	83%
newblue4	642k	637k	68	34%
newblue5	1228k	1284k	61	34%
newblue6	1248k	1288k	60	15%
newblue7	2481k	2637k	73	44%

TABLE V
RESULTING HPWLS FOR DIFFERENT CHIP UTILIZATIONS WITHOUT ("W/O") AND WITH THE MP-TREE ("MPT"). NR: NO LEGAL RESULTS CAN BE OBTAINED

	HPWL (×e7) by NTUplace3					
	utilization 85%		utilization 90%		utilization 95%	
Circuit	w/o	MPT	w/o	MPT	w/o	MPT
adaptec5	30.55	30.40	30.29	30.48	47.25	32.30
newblue1	6.64	6.30	6.74	6.38	6.85	6.62
newblue2	20.44	21.23	20.96	19.29	25.34	20.61
newblue3	NR	31.21	NR	29.64	NR	38.68
newblue4	22.82	21.41	26.70	22.68	26.83	23.77
newblue5	41.09	40.21	49.12	47.97	72.56	68.14
newblue6	45.45	45.46	53.14	47.60	66.51	65.21
newblue7	111.92	114.12	NR	120.15	NR	136.87
Comp.	1.00	0.99	1.00	0.93	1.00	0.88

A. *Effects of Chip Utilization Rates*

In this experiment, we used the ISPD'06 Placement Contest Benchmarks [18]. We changed all fixed macros in the benchmarks to movable ones to test our macro placement algorithm. Table IV shows the statistics of the ISPD'06 benchmarks. The cell numbers range from 842k to 2481k, and the macro numbers range from 9 to 73. "# Macros" gives the number of macros handled by the MP-tree. The area of these macros is larger than 1000 times of the average block area. "MA-ratio" gives the total area of those macros over the total area of all blocks. Since modern ASIC designs, such as consumer products, usually have high chip utilization rates to reduce the cost, we modified the core region in these benchmarks to obtain three different utilization rates—85%, 90%, and 95%. To show the effectiveness of the MP-tree, we compared NTUplace3 alone with NTUplace3 integrated with the MP-tree. Specifically, the MP-tree macro placer took NTUplace3's global placement results, optimized the macro positions, and fixed all macros. Then, the remaining cells are placed by NTUplace3 again. Table V shows the resulting HPWLs with different chip utilization rates. The columns "w/o" give the resulting HPWLs using NTUplace3 alone, whereas the columns "MPT" give the resulting HPWLs using NTUplace3 integrated with the MP-tree.

Integrating the MP-tree with NTUplace3, we can obtain legal placements with shorter HPWLs for most circuits; in contrast, NTUplace3 alone may not obtain legal placements for several benchmark circuits. In particular, the higher the chip utilization rate, the larger the average HPWL reduction. The average HPWL reductions are 7% and 12% for the 90% and 95% chip utilization rates, respectively.

The results also show that when the chip utilization is higher, NTUplace3 obtained longer HPWLs and failed to find legal placements on more circuits. The reason is that the macro positions are not guaranteed to be overlap-free in analytical placers, and, thus, it is harder to find legal solutions. Note that the circuit newblue3 has the highest MA-ratio, i.e., 83%, and NTUplace3 alone could not find any legal placement for this circuit. In contrast, NTUplace3 with the MP-tree robustly generated legal placements under different chip utilizations. Fig. 12(a) and (b) shows the resulting placements for newblue3 with 95% utilization using NTUplace3 alone and the MP-tree with NTUplace3, respectively.

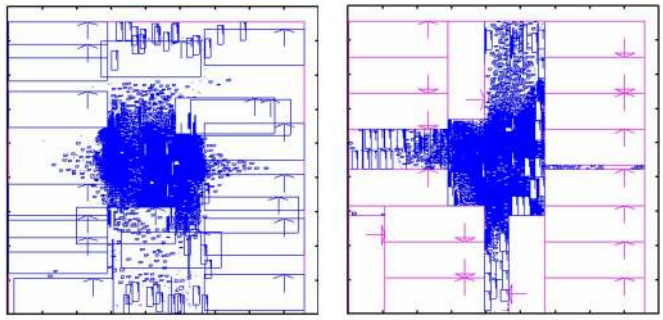


Fig. 12. (a) Illegal placement generated by NTUplace3 alone for the circuit newblue3. (b) Placement generated by NTUplace3 with the MP-tree for newblue3.

TABLE VI
RESULTING HPWLS OF USING THE MP-TREE AND FOUR PACKING TREES FOR THE MACRO PLACEMENT (UTILIZATION RATE = 90%). NR: NO LEGAL RESULTS CAN BE OBTAINED

	NTUplace3 + MP-tree		NTUplace3 + Packing Trees	
Circuit	HPWL (×e7)	CPU (min)	HPWL (×e7)	CPU (min)
adaptec5	30.48	76	32.17	89
newblue1	6.38	22	6.55	21
newblue2	19.29	35	NR	27
newblue3	29.64	98	31.52	103
newblue4	22.68	77	23.14	66
newblue5	47.97	315	61.13	290
newblue6	47.60	144	48.95	156
newblue7	120.15	729	134.06	1107
Comp.	1.00	1.00	1.08	1.09

B. *Comparison Between MP-Trees and Packing Trees*

This experiment studies the difference between the MP-tree and the independent four packing trees described in Section II-B. The method of using four packing trees is a simple extension to a macro placement for a rectangular chip. We divided a chip into four subregions and created four different BL-/BR-/TL-/TR-packing trees in the corresponding subregions. Note that, although this extension for packing trees still can handle the macro placement in a chip, it has many limitations; for example, it is much harder to deal with the region constraints that cross different regions.

We used the ISPD'06 benchmarks with the 90% chip utilization rate, and the results are shown in Table VI. The CPU time consists of the time for both macro and standard-cell placements; the macro placement takes only less than 1 min for an instance with hundreds of macros. From the results, we observed that the MP-tree is more robust in finding legal placements for all benchmarks, whereas the method with four packing trees cannot be robust. For those benchmarks with

TABLE VII
RESULTING HPWLS AND CPU TIMES FOR DIFFERENT PLACERS WITHOUT ("W/O") AND WITH MP-TREES ("MPT"; UTILIZATION RATE = 90%). NR: NO LEGAL RESULTS CAN BE OBTAINED

	Capo				mPL			
	HPWL (×e7)		CPU (min)		HPWL (×e7)		CPU (min)	
Circuit	w/o	MPT	w/o	MPT	w/o	MPT	w/o	MPT
adaptec5	38.29	33.52	432	537	NR	28.72	NR	138
newblue1	9.56	6.71	155	109	6.45	6.18	47	47
newblue2	25.99	22.05	287	234	NR	18.18	NR	94
newblue3	33.27	34.00	263	432	NR	31.11	NR	116
newblue4	26.93	24.00	311	451	NR	21.04	NR	93
newblue5	47.07	42.96	775	894	NR	39.94	NR	239
newblue6	55.22	49.23	795	882	NR	45.33	NR	296
newblue7	119.48	107.99	1795	2752	NR	94.76	NR	588
Comp.	1.00	0.88	1.00	1.21	1.00	0.96	1.00	0.99

legal placements, the MP-tree can further reduce the average HPWL by 8% under comparable running times, which shows the effectiveness of the MP-tree.

C. Integration With Other Placers

In addition to NTUplace3, we also integrated our MP-tree with Capo 10.2 and mPL6, which are based on the min-cut and analytical placement techniques, respectively. Table VII shows the results without and with the MP-tree based on the ISPD '06 benchmarks. Again, we used the 90% chip utilization rate for all circuits, and the CPU time for MPT includes macro and cell placements. Capo is robust in finding legal placements since macro positions are guaranteed to be overlap-free during the global placement. However, the quality is not good. Integrated with the MP-tree, Capo reduced the average HPWL by 12% than that without the MP-tree. We tried several times, but mPL alone could not obtain legal solutions for seven circuits. With the MP-tree, however, mPL can obtain legal solutions for all circuits. This shows that the MP-tree is robust in finding legal solutions.

D. Results on the IBM Mixed-Size Benchmarks

We also tested our MP-tree on the IBM-Dragon mixed-size benchmark suite [16]. See Table VIII for the number of large macros and the resulting HPWLs for Capo and NTUplace based on this benchmark suite. As shown in this table, this benchmark suite contains only a few large macros, which is very different from the ISPD '06 one. Despite the circuit properties, the MP-tree can still outperform Capo by 5% in the average HPWL and obtain comparable results with NTUplace. Compared with the ISPD '06 benchmarks, we observe that the IBM-Dragon mixed-size benchmarks are much easier and can reasonably be handled by some existing placers; therefore, they are not of our main interest for using the MP-tree, which is intended for modern mixed-size designs with large macros and high chip utilization rates. Note that the two circuits—ibm01 and ibm05—do not contain large macros and are, thus, not shown in the table.

E. Routing Results on the Faraday Benchmarks

Table IX lists the statistics of the Faraday benchmarks [16], [17]. Note that the direct-memory-access circuit is not used in our experiment since it has no macro. There are two (seven) macros in each of the DSP (RISC) circuits. The macro area

TABLE VIII
RESULTING HPWLS FOR DIFFERENT PLACERS WITHOUT ("W/O") AND WITH MP-TREES ("MPT") FOR THE IBM-DRAGON MIXED-SIZE BENCHMARKS (UTILIZATION RATE = 90%)

		HPWL			
	Large	Capo		NTUplace	
Circuit	Macro #	w/o	MPT	w/o	MPT
ibm02	4	5.80 e6	5.62 e6	5.03 e6	5.00 e6
ibm03	2	7.89 e6	7.82 e6	6.84 e6	7.14 e6
ibm04	3	9.67 e6	8.80 e6	7.76 e6	7.66 e6
ibm06	2	7.47 e6	7.42 e6	6.08 e6	6.14 e6
ibm07	7	1.29 e7	1.13 e7	1.00 e7	1.02 e7
ibm08	4	1.46 e7	1.40 e7	1.29 e7	1.26 e7
ibm09	7	1.40 e7	1.44 e7	1.19 e7	1.20 e7
ibm10	8	3.77 e7	3.73 e7	3.04 e7	3.00 e7
ibm11	8	2.11 e7	2.01 e7	1.79 e7	1.79 e7
ibm12	6	4.10 e7	3.51 e7	3.30 e7	3.38 e7
ibm13	12	2.76 e7	2.50 e7	2.26 e7	2.28 e7
ibm14	6	3.92 e7	3.83 e7	3.58 e7	3.58 e7
ibm15	4	5.19 e7	5.17 e7	4.78 e7	4.70 e7
ibm16	27	6.12 e7	5.94 e7	5.86 e7	5.98 e7
ibm17	6	7.24 e7	7.46 e7	6.66 e7	6.59 e7
ibm18	5	4.49 e7	4.45 e7	4.19 e7	4.31 e7
Comp.		1.00	0.95	1.00	1.00

TABLE IX
STATISTICS OF FARADAY BENCHMARKS

Circuit	# Cells	# Nets	Row-Util	# Macros	MA-ratio
DSP1	26299	28447	91%	2	22%
DSP2	26279	28431	90%	2	7%
RISC1	32615	34034	94%	7	42%
RISC2	32615	34034	94%	7	38%

ranges from 6.96% to 41.99% of the whole chip area in these benchmarks. It will be clear later that most existing placers cannot handle these circuits well even when there are only a few macros.

Table X gives the mixed-size placement and routing results for Fengshui, mPL, Capo, our MP-tree macro placer integrated with Capo, APlace, the MP-tree integrated with APlace, NTUplace3, and the MP-tree integrated with NTUplace3 on the Faraday benchmarks. We do not integrate the MP-tree with Fengshui and mPL because both placers cannot correctly handle the MP-tree's preplaced macros. In the table, "HPWL" and "WL" (the routed wirelength) are reported in the database unit, and "Viol" gives the number of violations in the routing solutions. Our MP-tree macro placer takes Capo's results as the initial macro positions, which needs only a few seconds for these benchmarks since the numbers of macros are small. Therefore, the runtimes for the macro placement alone are only a few seconds and not reported.

As shown in the table, the min-cut placer Fengshui generates the results with many macros/cells outside the chip region (same as observed in [17]). Also, APlace (NTUplace3) alone generated many overlaps for three circuits (one circuit) and failed to legalize them (it); for these illegal detailed placement results, we only report the HPWLs of their global placement solutions. The min-cut floorplacer Capo and the analytical placer mPL can find legal solutions for all Faraday benchmarks; however, Capo achieves much better HPWLs than mPL. Considering only those with legalized detailed placement solutions, APlace achieves the best HPWL and routed WL, followed by NTUplace3, Capo, Fengshui, and mPL (mPL obtains a better WL than Fengshui).

TABLE X
PLACEMENT AND ROUTING RESULTS FOR THE FARADAY BENCHMARKS. THE WIRELENGTH IS IN THE DATABASE UNIT. VIOL IS THE NUMBER OF VIOLATIONS FOR THE ROUTING RESULTS

Circuit	Fengshui					mPL				
	Place		Route			Place		Route		
	HPWL (×e8)	Time (min)	WL (×e8)	Time (min)	Viol	HPWL (×e8)	Time (min)	WL (×e8)	Time (min)	Viol
DSP1	*(13.25)	6	NR	NR	NR	13.41	4	18.69	14	8998
DSP2	9.08	6	12.10	8	0	11.22	4	14.87	13	1
RISC1	*(18.53)	17	NR	NR	NR	24.92	8	36.60	70	99613
RISC2	19.10	17	45.10	66	452321	23.90	10	33.50	19	29682
Avg.	1.35		1.68			1.63		1.62		

Circuit	Capo					MP-tree + Capo				
	Place		Route			Place		Route		
	HPWL (×e8)	Time (min)	WL (×e8)	Time (min)	Viol	HPWL (×e8)	Time (min)	WL (×e8)	Time (min)	Viol
DSP1	10.09	8	12.70	9	1	9.32	7	12.06	7	0
DSP2	8.91	8	11.37	8	0	8.98	7	11.50	7	0
RISC1	16.35	16	25.70	32	265	14.63	12	21.54	25	6
RISC2	16.02	14	23.75	22	6	14.04	12	19.5	13	2
Avg.	1.15		1.15			1.07		1.03		

Circuit	NTUplace3					MP-tree + NTUplace3				
	Global Place		Route			Place		Route		
	HPWL (×e8)	Time (min)	WL (×e8)	Time (min)	Viol	HPWL (×e8)	Time (min)	WL (×e8)	Time (min)	Viol
DSP1	9.31	3	11.86	9	0	8.98	2	11.72	9	0
DSP2	8.97	2	11.44	7	0	8.61	2	11.35	8	0
RISC1	*(12.80)	5	NR	NR	NR	13.78	5	22.06	18	1
RISC2	15.65	6	20.85	10	0	13.72	6	19.25	19	0
Avg.	*(1.06)		1.04			1.02		1.03		

Circuit	APlace					MP-tree + APlace				
	Global Place		Route			Place		Route		
	HPWL (×e8)	Time (min)	WL (×e8)	Time (min)	Viol	HPWL (×e8)	Time (min)	WL (×e8)	Time (min)	Viol
DSP1	*(9.04)	20	NR	NR	NR	8.88	13	11.57	8	1
DSP2	8.69	11	11.20	8	0	8.65	12	11.12	8	0
RISC1	*(13.07)	22	NR	NR	NR	13.12	25	19.96	25	0
RISC2	*(13.80)	22	NR	NR	NR	13.27	21	19.87	24	0
Avg.	*(1.01)		1.01			1.00		1.00		

*: THE PLACEMENT RESULT HAS A LOT OF OVERLAPS OR MANY BLOCKS ARE OUTSIDE THE PLACEMENT REGION AND FAILED TO LEGALIZE. NR: NO RESULT DUE TO NO LEGAL PLACEMENT.

However, APlace failed to generate legal solutions for three out of four circuits, although there are only two or seven macros in the circuits; the mixed-size placer Fengshui had the same problem and failed to generate legal solutions for two circuits. The results of APlace and Fengshui confirm the drawbacks of the first-type mixed-size placers mentioned in Section I-A.

In contrast, our two-stage mixed-size placement approach can find legal placement solutions for all the circuits. Our MP-tree macro placer integrated with Capo (as the global and standard-cell placer) can reduce the HPWL and the routed WL by 8% and 12%, respectively, on average, compared with Capo alone. Similarly, our MP-tree integrated with NTUplace3 can generate a feasible placement for all circuits, and the average HPWL and WL are 13% and 12% shorter than those of Capo, respectively. In particular, our MP-tree integrated with APlace can generate feasible placements for all circuits, and the quality is superior to all other combinations. The HPWLs are reduced by 35%, 63%, 15%, and 6% compared with Fengshui, mPL, Capo, and NTUplace3, respectively. Furthermore, the routed WLs are 68%, 62%, 15%, and 4% better than Fengshui, mPL, Capo, and NTUplace3, respectively.

Although Capo's floor-placement approach can find legal solutions for all the four circuits, we observe that Capo cannot handle a macro placement well since our MP-tree macro placer with Capo can reduce the HPWL and the WL by 10%, on average, compared to Capo alone. Furthermore, the MP-tree can save the CPU time for Capo since Capo does not need extra time to find legal macro positions. From Table X, we further observe the fact that the larger the total macro area is, the more HPWL reduction our placement flow can achieve. Table XI summarizes the WL reductions. In particular, integrating with the MP-tree can also significantly reduce the number of violations during routing. The results show the effectiveness of our MP-tree macro placer. Fig. 13 shows the placement results using the MP-tree integrated with APlace 2.0.

F. Routing Results on the mchip Circuits

In this experiment, we further show that the macro placements generated by the MP-tree lead to shorter HPWLs and better routability based on four real industry circuits for cell phones, digital video disk players, personal digital assistants, etc. Table XII lists the statistics. The numbers of cells range

TABLE XI
SUMMARIES OF THE WIRELENGTH REDUCTION ON THE FARADAY BENCHMARKS

Circuit	Macro Area	Normalized HPWL				Normalized WL			
		Capo	MP-tree + Capo	MP-tree + NTUplace3	MP-tree + APlace	Capo	MP-tree + Capo	MP-tree + NTUplace3	MP-tree + APlace
DSP2	6.96%	1.00	1.01	0.97	0.97	1.00	1.01	1.00	0.98
DSP1	21.98%	1.00	0.92	0.89	0.88	1.00	0.95	0.92	0.91
RISC2	37.37%	1.00	0.88	0.86	0.83	1.00	0.82	0.81	0.84
RISC1	41.99%	1.00	0.89	0.84	0.80	1.00	0.84	0.86	0.78

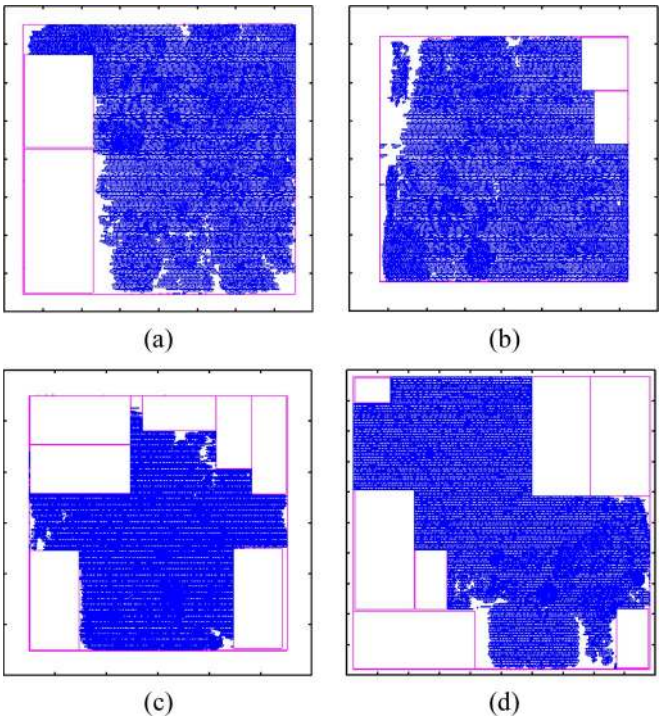


Fig. 13. Layouts for the Faraday benchmarks generated by our MP-tree macro placer combined with APlace 2.0 for a standard-cell placement. Large rectangles are the macros. (a) DSP1. (b) DSP2. (c) RISC1. (d) RISC2.

TABLE XII
STATISTICS OF THE MCHIP CIRCUITS

Circuit	# Cells	# Nets	Row-Util	# Macros	MA-ratio
mchip1	540k	570k	94%	50	66%
mchip2	820k	860k	91%	95	56%
mchip3	910k	960k	88%	110	54%
mchip4	1320k	1300k	90%	380	36%

from 540k to 1320k, and the numbers of macros range from 50 to 380. There are also some preplaced blocks and some blockages in these circuits. From the previous experiments, we found that only Capo can find legal results for the mixed-size placement with large macros and can correctly handle preplaced blocks and blockages. As mentioned in the classification of the mixed-size placement methods in Section I-A, the first type might not generate overlap-free macro positions. Therefore, we compared the MP-tree-based macro placement (the third type) with that of Capo (the second type). That is, our MP-tree and Capo are used to place macros only, and the remaining cells are placed by Synopsys Astro. After the placement, we performed the global routing.

Table XIII shows the resulting HPWLs, routed WLs, global routing cell (GRC) overflows, and max overflows. The GRC overflow is the percentage of the GRCs that have overflows. The larger the value, the more congested the placement. "Max overflow" gives the number of extra tracks assigned for the GRC with the maximum overflow.

For the four circuits, our MP-tree consistently obtains much better WLs (an HPWL and a WL) than Capo's macro placement. Furthermore, Capo's macro placement results in larger GRC overflows and max overflows, and requires greater runtime for the cell placement and routing than the MP-tree. The results show that Capo's macro placement results in more congested placements than the MP-tree. Specifically, Capo's average HPWL and average routed WL are about 35% and 55% longer than those of the MP-tree, respectively. These results show that the macro placements generated by the MP-tree have better routability. In particular, the MP-tree leads to much better efficiency.

Fig. 14(a) shows the macro placement result of mchip2, which contains 95 macros by using the MP-tree. Fig. 14(b) shows the placement of mchip4 with 380 macros and four region constraints.

VI. CONCLUSION

We have proposed a novel macro placement algorithm based on the MP-tree representation. Experimental results have shown that our algorithm is robust in finding legal macro placements and routable results, and can obtain much smaller WLs than leading academic mixed-size placers alone, based on benchmarks with large macros and high utilization rates. Integrating our macro placer with the leading academic standard-cell placers, such as APlace 2.0, Capo 10.2, mPL6, and NTUplace3, we can easily find legal mixed-size placement results with significantly better WLs and routability. Our studies have shown that the MP-tree can be adopted as a niche tool that performs particularly well for the "difficult" instances with large macros and high utilization rates. In particular, the MP-tree is also complementary to other leading placers and can be combined with them to form a comprehensive placer for general mixed-size designs.

APPENDIX I
MACRO PLACEMENT CONSTRAINTS

With the nice properties of the MP-tree and the binary tree, the MP-tree can easily handle various placement constraints.

RECTILINEAR PLACEMENT REGION

In the hierarchical design methodology, a chip is divided into several partitions. These partitions are usually rectilinear instead of square. Since the MP-tree packs macros to corners,

TABLE XIII
COMPARISON OF CAPO'S MACRO PLACER AND OUR MP-TREE MACRO PLACER

Circuit	Capo's Macro Placement + Commercial Std-Cell Placer						MP-tree + Commercial Std-Cell Placer					
	Place		Route				Place		Route			
	HPWL (×e7)	Time (min)	WL (×e7)	Time (min)	GRC Overflow	Max Overflow	HPWL (×e7)	Time (min)	WL (×e7)	Time (min)	GRC Overflow	Max Overflow
mchip1	5.84	16	6.56	23	0.7%	39	5.26	8	6.13	7	0.7%	5
mchip2	5.65	28	6.65	32	1.0%	27	4.72	13	5.34	8	0.1%	4
mchip3	10.00	23	16.90	180	36.4%	113	5.26	16	6.02	14	0.1%	4
mchip4	14.12	41	14.16	323	1.4%	288	11.76	31	13.27	45	0.1%	31
Average	1.35	1.73	1.55	6.83	97.25	13.02	1.00	1.00	1.00	1.00	1.00	1.00

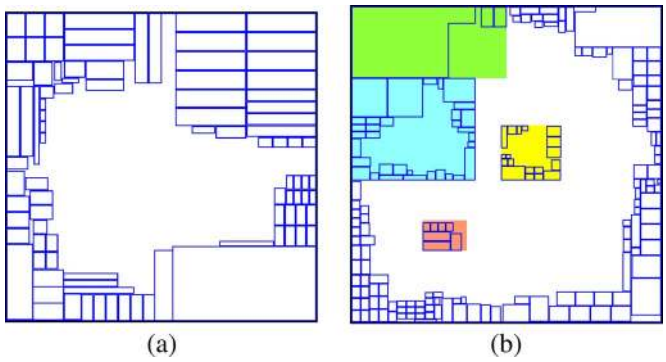


Fig. 14. (a) Macro placement of mchip2 with 95 macros by using the MP-tree. (b) Macro placement of mchip4 with 380 macros and four region constraints.

it is very easy to handle the rectilinear placement region by creating one packing subtree for each convex corner.

PLACEMENT BLOCKAGES AND PREPLACED MACROS

The placement blockages are given by users, and no macro can overlap with any blockage. During packing, we place each macro and check if it overlaps with a blockage. If it does, we shift the macro horizontally or vertically to the nearest position with no overlap. A preplaced macro can be treated as a placement blockage, and there is no need to add such a corresponding node to the MP-tree.

CORNER MACROS

Some macros, such as analog blocks, are usually fixed at a corner; we call them *corner macros*. We fix the node corresponding to the corner macro as the root of the packing subtree. Thus, the corner macro can be fixed at the corresponding corner.

RECTILINEAR MACROS

We adopt the method proposed in [29] to handle rectilinear macros for our MP-tree. A rectilinear macro is sliced into several rectangular macros. The *location constraint* corresponding to the tree topology is created. During packing, we shift a *misalignment* macro upward to maintain the rectilinear shape.

MACRO CLUSTERING AND PERFORMANCE CONSTRAINTS

Our macro placer considers the design hierarchy to cluster macros to reduce the problem size. The macros in the same group of the design hierarchy will be clustered if they have the same height/width. These macros usually have strong correlations, and, thus, clustering macros not only utilizes the area

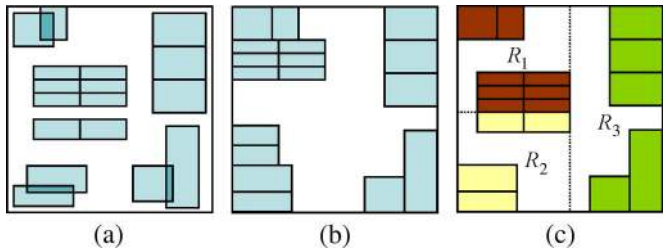


Fig. 15. Example of region constraints. (a) Initial macro placement. (b) Macro placement using an MP-tree without region constraints. (c) Macro placement using an MP-tree with region constraints. There are three regions, i.e., R_1, R_2, and R_3, which contain eight, four, and five macros, respectively. The macro displacement in (c) is smaller than that in (b).

better but also places strongly correlated macros closer. We shall consider only the cluster dimensions that do not produce any waste area. For example, for the clustering of four macros, it has three possible cluster matrices, i.e., 1×4, 2×2, and 4×1. The desired clustering dimension is selected during simulated annealing. For some macros, the timing between them is critical; we may also cluster these macros to satisfy the performance constraint. When declustering, the blocks are placed according to the current cluster matrix.

REGION CONSTRAINTS

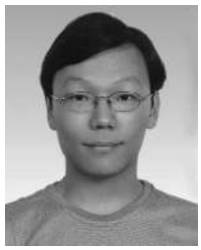
In a hierarchical design, a floorplan may be given. Based on the given floorplan, we can impose region constraints to macros so that these macros can only be placed into the corresponding regions. For each region, we create four packing subtrees for its four corners so that macros can be packed along the region boundary. Fig. 15 gives an example of the region constraints. Fig. 15(b) shows a macro placement from an MP-tree without any region constraints. If the regions and the corresponding macros are given, we may obtain a macro placement shown in Fig. 15(c) that has a smaller macro displacement. The region constraints are also important to timing-critical macros since they can keep these macros in the user-specified regions. In particular, the region constraints can also be used to reduce the macro displacement when the chip utilization is low.

REFERENCES

[1] T.-C. Chen, P.-H. Yuh, Y.-W. Chang, F.-J. Huang, and D. Liu, "MP-trees: A packing-based macro placement algorithm for mixed-size designs," in *Proc. ACM/IEEE Des. Autom. Conf.*, San Diego, CA, Jun. 2007, pp. 447–452.

[2] E. Wein and J. Benkoski, "Hard macros will revolutionize SoC design," *EETimes Online*, Aug. 2004. [Online]. Available: http://www.eetimes.com/showArticle.jhtml?articleID=26807055

[3] T. Taghavi, X. Yang, B.-K. Choi, M. Wang, and M. Sarrafzadeh, "Dragon2006: Blockage-aware congestion-controlling mixed-size placer," in *Proc. ACM Int. Symp. Phys. Des.*, San Jose, CA, Apr. 2006, pp. 209–211.
[4] M. Upton, K. Samii, and S. Sugiyama, "Integrated placement for mixed macro cell and standard cell designs," in *Proc. ACM/IEEE Des. Autom. Conf.*, Orlando, FL, Jun. 1990, pp. 32–35.
[5] C.-C. Chang, J. Cong, and X. Yuan, "Multi-level placement for large-scale mixed-size IC designs," in *Proc. IEEE/ACM Asia South Pacific Des. Autom. Conf.*, Kitakyushu, Japan, Jan. 2003, pp. 325–330.
[6] A. R. Agnihotri, S. Ono, C. Li, M. C. Yildiz, A. Khatkhate, C.-K. Koh, and P. H. Madden, "Mixed block placement via fractional cut recursive bisection," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 5, pp. 748–761, May 2005.
[7] T.-C. Chen, T.-C. Hsu, Z.-W. Jiang, and Y.-W. Chang, "NTUplace: A ratio partitioning based placement algorithm for large-scale mixed-size designs," in *Proc. ACM Int. Symp. Phys. Des.*, San Francisco, CA, Apr. 2005, pp. 236–238.
[8] A. B. Kahng and Q. Wang, "A faster implementation of APlace," in *Proc. ACM Int. Symp. Phys. Des.*, San Jose, CA, Apr. 2006, pp. 218–220.
[9] P. Spindler and F. M. Johannes, "Fast and robust quadratic placement combined with an exact linear net model," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, San Jose, CA, Nov. 2006, pp. 179–186.
[10] T. Chan, J. Cong, J. Shinnerl, K. Sze, and M. Xie, "mPL6: Enhanced multilevel mixed-size placement," in *Proc. ACM Int. Symp. Phys. Des.*, San Jose, CA, Apr. 2006, pp. 212–214.
[11] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, and Y.-W. Chang, "NTUplace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 7, pp. 1228–1240, Jul. 2008.
[12] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, and Y.-W. Chang, "A high-quality mixed-size analytical placer considering preplaced blocks and density constraints," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, San Jose, CA, Nov. 2006, pp. 187–192.
[13] J. Roy, D. Papa, A. Ng, and I. Markov, "Satisfying whitespace requirements in top-down placement," in *Proc. ACM Int. Symp. Phys. Des.*, San Jose, CA, Apr. 2006, pp. 206–208.
[14] S. N. Adya and I. L. Markov, "Combinatorial techniques for mixed-size placement," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 10, no. 5, pp. 58–90, Jan. 2005.
[15] R. Auletta, "Expert system perimeter block placement floorplanning," in *Proc. IEEE/ACM Des. Autom. Test Eur. Conf.*, Paris, France, Feb. 2004, pp. 140–143.
[16] *ICCAD04 Mixed-Size Placement Benchmarks*. [Online]. Available: http://vlsicad.eecs.umich.edu/BK/ICCAD04bench/
[17] S. N. Adya, S. Chaturvedi, J. A. Roy, D. A. Papa, and I. L. Markov, "Unification of partitioning, placement and floorplanning," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, San Jose, CA, Nov. 2004, pp. 550–557.
[18] *ISPD 2006 Placement Contest*. [Online]. Available: http://www.sigda.org/ispd2006/contest.html
[19] Y.-C. Chang, Y.-W. Chang, G.-M. Wu, and S.-W. Wu, "B*-trees: A new representation for non-slicing floorplans," in *Proc. ACM/IEEE Des. Autom. Conf.*, Los Angeles, CA, Jun. 2000, pp. 458–463.
[20] N. Viswanathan and C. C.-N. Chu, "FastPlace: Efficient analytical placement using cell shifting, iterative local refinement and a hybrid net model," in *Proc. ACM Int. Symp. Phys. Des.*, Apr. 2004, pp. 26–33.
[21] P. Hilton and J. Pederson, "Catalan numbers, their generalization, and their uses," *Math. Intell.*, vol. 13, no. 2, pp. 64–75, 1991.
[22] K. Sakanushi and Y. Kajitani, "The quarter-state sequence (Q-sequence) to represent the floorplan and applications to layout optimization," in *Proc. IEEE Asia-Pacific Conf. Circuits Syst.*, 2000, pp. 829–832.
[23] D. F. Wong and C. L. Liu, "A new algorithm for floorplan design," in *Proc. ACM/IEEE Des. Autom. Conf.*, 1986, pp. 101–107.
[24] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, "Rectangle-packing-based module placement," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, San Jose, CA, Nov. 1995, pp. 472–479.
[25] J.-M. Lin and Y.-W. Chang, "TCG: A transitive closure graph-based representation for general floorplans," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 13, no. 2, pp. 288–292, Feb. 2005.
[26] J.-M. Lin and Y.-W. Chang, "TCG-S: Orthogonal coupling of P*-admissible representations for general floorplans," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 23, no. 6, pp. 968–980, Jun. 2004.
[27] A. B. Kahng, S. Reda, and Q. Wang, "Architecture and details of a high quality, large-scale analytical placer," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, San Jose, CA, Nov. 2005, pp. 890–897.
[28] A. R. Agnihotri, S. Ono, and P. H. Madden, "Recursive bisection placement: Feng Shui 5.0 implementation details," in *Proc. ACM Int. Symp. Phys. Des.*, San Francisco, CA, Apr. 2005, pp. 230–232.
[29] G.-M. Wu, Y.-C. Chang, and Y.-W. Chang, "Rectilinear block placement using B*-trees," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 8, no. 2, pp. 188–202, Apr. 2003.



Tung-Chieh Chen (S'04) received the B.S. degree in electrical engineering and the Ph.D. degree in electronics engineering from the National Taiwan University, Taipei, Taiwan, R.O.C., in 2003 and 2008, respectively.

In 2007, he was a Visiting Ph.D. Student with The University of Texas, Austin. He is currently with the Graduate Institute of Electronics Engineering, National Taiwan University.

Dr. Chen was the recipient of the first prize of the 2007 Association for Computing Machinery (ACM) Special Interest Group on Design Automation CADathlon programming contest and the third place of the 2006 ACM International Symposium on Physical Design Placement Contest.

Ping-Hung Yuh received the B.S. and Ph.D. degrees in computer science and information engineering from the National Chiao Tung University, Hsinchu, Taiwan, R.O.C., in 2002, and the National Taiwan University, Taipei, Taiwan, in 2008, respectively.

His current research interests include temporal floorplanning for reconfigurable computing and biochip design automation, including placement and routing.

Yao-Wen Chang (S'94–A'96–M'96) received the B.S. degree from National Taiwan University (NTU), Taipei, Taiwan, in 1988, and the M.S. and Ph.D. degrees from the University of Texas at Austin in 1993 and 1996, respectively, all in computer science.

He is a Professor in the Department of Electrical Engineering and the Graduate Institute of Electronics Engineering, NTU. He is currently also a Visiting Professor at Waseda University, Kitakyushu, Japan. His current research interests lie in VLSI physical design and design for manufacturability/reliability.

Fwu-Juh Huang received the M.S. degree in computer engineering from the Syracuse University, Syracuse, NY, in 2001.

Since 2004, he has been a Senior Engineer with MediaTek Inc., Hsinchu, Taiwan, R.O.C., where his main work is involved with the physical integration for low-power chip and automatic place-and-route flow automation.

Tien-Yueh Liu received the B.S. and M.S. degrees in computer science engineering from the National Tsing Hua University, Hsinchu, Taiwan, R.O.C., in 1996 and 1998, respectively.

He is currently with MediaTek Inc., Hsinchu, where his work focuses on physical design flow and implementation.