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

Tung-Chieh Chen[1], Ping-Hung Yuh[1], Yao-Wen Chang[1,2],
Fwu-Juh Huang[3] and Denny Liu[3]
[1]Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 106, Taiwan
[2]Department of Electrical Engineering, National Taiwan University, Taipei 106, Taiwan
[3]MediaTek, Inc., Hsin-Chu 300, Taiwan
donnie@eda.ee.ntu.edu.tw; r91089@csie.ntu.edu.tw; ywchang@cc.ee.ntu.edu.tw;
fj_huang@mtk.com.tw; denny_liu@mtk.com.tw

ABSTRACT

In this paper, we present a new multi-packing tree (MP-tree) representation for macro placement to handle mixed-size designs. Based on binary trees, the MP-tree is very efficient, effective, and flexible for handling macro placement with various constraints. Given a global placement, 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 eight ISPD'06 placement contest benchmarks show that our macro placer combined with Capo 10.2, NTUplace3, or mPL6 for standard-cell placement outperforms these state-of-the-art academic mixed-size placers alone by large margins in both robustness and quality. In addition to wirelength, experimented on five real industrial designs show that our method significantly reduce the average HPWL by 35%, the average routed wirelength by 55%, and the routing overflows than the counterpart with Capo 10.2, implying that our macro placer leads to much higher routability.

Categories and Subject Descriptors: B.7.2 [Integrated Circuits]: Design Aids - Layout, Placement and Routing

General Terms: Algorithms, Design

Keywords: Macro placement, Multi-packing tree

1. INTRODUCTION

Due to the use of IP (Intellectual Property) modules and embedded memories, a modern VLSI chip often consists of a significant number of macros. Mixed-size placement, which places both macros and standard cells, becomes more and more popular for real-world applications. In particular, the number of macros in a modern system-on-chip (SOC) design is expected to increase dramatically in the future [14]. As a result, many mixed-size placement flows/algorithms are proposed in the literature [2–5, 7–12] recently.

1.1 Previous Work

According to the macro handling methods, we can classify the mixed-size placement algorithms into three types. The first type places macros and standard cells simultaneously. A 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. Especially when the chip utilization rate is high, it is possible that no legal solution can be obtained. The simulated-annealing-based placers, Dragon [12] and mPG-MS [5], the min-cut-based placers, Feng Shui [3] and NTUplace [7], and the analytical placers, APlace [9], Kraftwerk [11], mPL [4], and NTUplace3 [8], belong to this type.

The second type places macros constructively. 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 may be longer than those from the first type. The min-cut floorplacer, Capo [10], belongs to this type. The 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, macro placement and then standard-cell placement. The macro positions are first determined, and then standard cells are placed into the rest area. A combinatorial technique was proposed in [2]. It 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. Then, macros are fixed and standard cells are re-placed in the remaining whitespace using a standard-cell placer. Compared with the previous 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. Further, macro orientations and placement constraints, such as pre-placed macros and placement blockages, can be handled more easily. Thus, the two-stage approach is widely used in the industry.

Because of the advantages of the third type placers, we adopt the two-stage mixed-size placement approach. Our work focuses on the first stage, macro placement, which is crucial for mixed-size placement since macro positions significantly affect the placement of standard cells and the final placement and routing quality.

1.2 Our Contributions

In this paper, we present a new *multi-packing tree* (*MP-tree* for short) representation for macro placement in mixed-size designs. Given an initial global placement, the MP-tree-based macro placer optimizes macro positions to remove overlaps, minimize the macro displacement from the initial macro positions, and maximizes the area of the chip center for standard-cell placement and routing. We summarize the advantages of the MP-tree-based macro placement algorithm as follows.

- 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.
- 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*). Because of the branch structure in the MP-tree, 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 8% shorter average wirelength than that

DAC 2007, June 4–8, 2007, San Diego, California, USA.

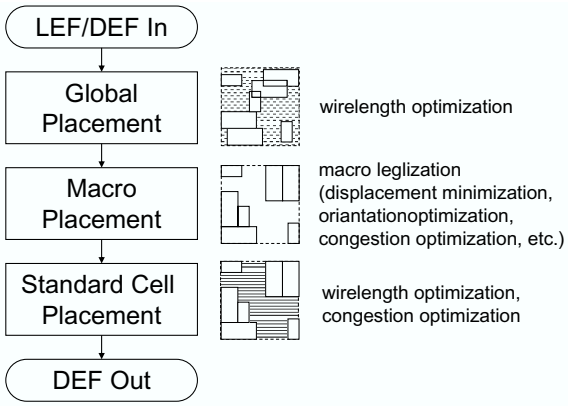


Figure 1: Our mixed-size placement flow.

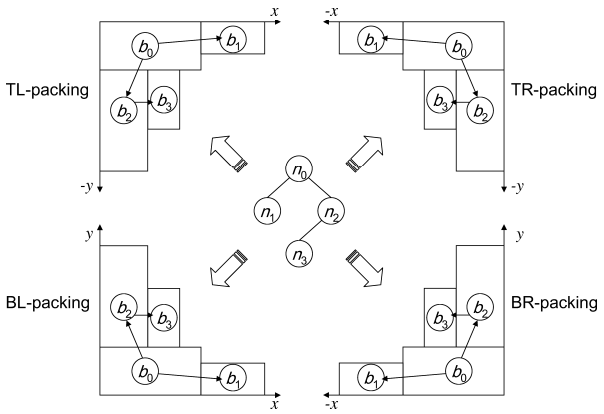


Figure 2: A packing tree with its four types of packing.

obtained by the traditional hierarchical method using four independent packing trees.

- The MP-tree combined with leading academic placers can robustly generate feasible results even for the designs with 95% chip utilization, while the leading mixed-size placers sometimes failed to find a legal result. Further, the placers with the MP-tree result in significantly shorter wirelength.
- Our two-stage mixed-size placement methodology is robust to find a desirable result. By minimizing macro displacement during the macro placement process, we can keep smaller resulting wirelength. By reserving chip center for standard-cell placement and routing, our macro placement has better routability and shorter routed wirelength.
- The MP-tree can easily handle various placement constraints commonly seen in real industry designs, such as pre-placed blocks, corner blocks, placement blockages, and region constraints. Thus, our macro placer is very flexible for practical applications.
- In addition to placement, we also experimented on five real industrial designs up to the routing stage. Compared with Capo 10.2's macro placer with a state-of-the-art commercial standard-cell placer, our MP-tree macro placer with the same standard-cell placer can fix one illegal design and further significantly reduce the average HPWL by 35%, the average routed wirelength by 55%, and the routing overflows for the other four legal designs, implying that our macro placer leads to much higher routability.

The rest of this paper is organized as follows. Section 2 introduces our mixed-size placement flow. Section 3 presents our macro placement algorithm. Section 4 gives the experimental results. Finally, the conclusion is given in Section 5.

2. MIXED-SIZE PLACEMENT

We adopt the two-stage mixed-size placement flow: (1) macro placement followed by (2) standard-cell placement. See Figure 1 for our mixed-size placement flow. After the circuit information is imported, a wirelength-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 in the chip region. With the objective of the macro displacement minimization, our macro placement algorithm results in better macro positions that lead to smaller wirelength in the later stage implicitly. The macro placement step plays an important role in the mixed-size placement. Finally, all macros are fixed and a standard-cell placer is then applied to place all standard cells in the available space.

Our macro placement is based on a packing technique so we can easily find legal solutions even for the cases with large macros and/or a large number of macros. Also, many real-world constraints can easily be considered; for example, the macro orientations can be optimized and the spacing between macros for routing can be reserved, which are important for practical applications.

3. MACRO PLACEMENT

Our design strategy is to pack macros inside the given region and to reserve the center region for standard-cell placement and routing. Since macros are usually large and there are routing blockages above macros in real-world applications, the macros tend to block the routes if they are placed in the chip center. Further, by minimizing the macro displacement, we can keep the desired wirelength implicitly since the given global placement is optimized for wirelength.

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

3.1 Packing-Tree Floorplan Representation

A packing tree is a binary tree for modeling non-slicing 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 packs the blocks to the bottom-left, top-left, top-right, and bottom-right corners, respectively.

Figure 2 shows a packing tree and its corresponding four packing types. Let (x_{corner}, y_{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

- (x_{corner}, y_{corner}) for BL-packing,
- $(x_{corner}, y_{corner} - h_{root})$ for TL-packing,
- $(x_{corner} - w_{root}, y_{corner} - h_{root})$ for TR-packing, and
- $(x_{corner} - w_{root}, y_{corner})$ for BR-packing.

If node n_j is the right child of n_i, the block b_j is

- the lowest adjacent block on the right with $x_j = x_i + w_i$ for BL-packing,
- the highest adjacent block on the right with $x_j = x_i + w_i$ for TL-packing,
- the highest adjacent block on the left with $x_j = x_i - w_j$ for TR-packing, and
- 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

- the first block above b_i with $x_j = x_i$ for BL-packing,
- the first block below b_i with $x_j = x_i$ for TL-packing,
- the first block below b_i with $x_j = x_i + w_i - w_j$ for TR-packing, and
- the first block above b_i with $x_j = x_i + w_i - w_j$ for BR-packing.

Therefore, given a packing tree, the x-coordinates of all blocks can be determined by traversing the tree once in linear time. Further, a y-coordinate can be computed using the contour data structure in amortized constant time, similar to the method used in [6]. So the complexity of transforming a packing-tree to the corresponding placement is amortized linear time. Note that B*-tree floorplan representation [6] is a BL-type packing 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 macro placement with placement constraints.

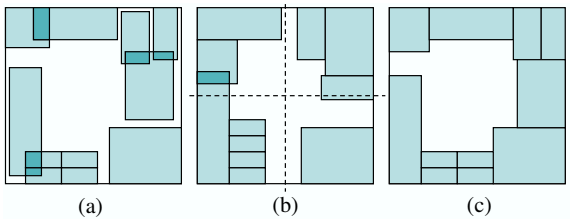


Figure 3: (a) An initial macro placement. (b) A macro placement by dividing the chip into four regions. The macros in the four regions are placed independently, and thus it lacks the global view of the placement interactions among different regions. (c) A better macro placement without dividing the chip into sub-regions.

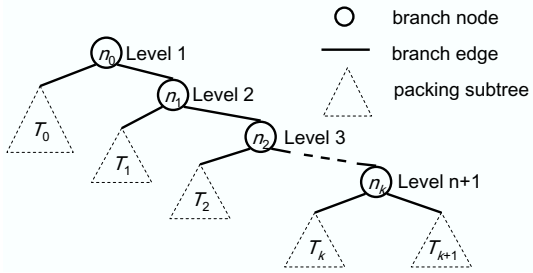


Figure 4: A general MP-tree.

3.2 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 over constrained because there is no real boundary between the regions. Further, 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.

Figure 3 illustrates the disadvantages of solving the macro placement problem by subdividing the chip. Given an initial macro placement in Figure 3(a), Figure 3(b) shows a possible floorplan result by dividing the chip into four sub-regions 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 large macro displacement. Further, 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. Figure 3(c) gives an example better solution with smaller macro displacement by employing the global optimization technique.

To implement the global optimization idea, we resort to the multi-packing tree (MP-tree) to handle the global interaction among different regions. An MP-tree combines several packing trees for different corners. Figure 4 shows an example 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, the earlier the packing subtree packs blocks. If the parent of two packing subtrees is 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. For example, Figure 5 shows an MP-tree with five packing subtrees for an L-shaped rectilinear placement region. The packing subtree T_i packs blocks to the corner c_i.

The MP-tree structure directly induces a special hierarchical framework for the optimization of macro placement. Each packing subtree handles macros being packed to a corner, i.e., perform *local optimization*. A major drawback of the traditional

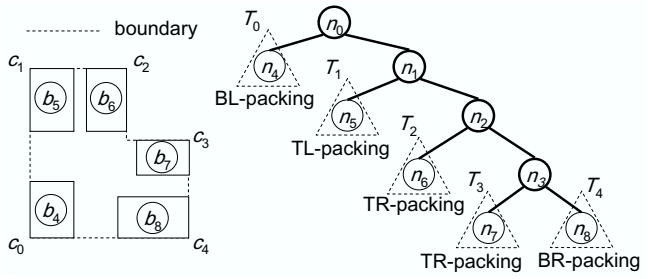


Figure 5: An MP-tree with five packing subtrees for an L-shaped rectilinear placement region. The packing subtree T_i packs blocks to the corner c_i.

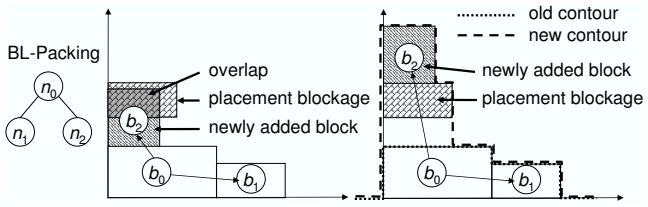


Figure 6: The placement blockage is handled during packing. After adding a new macro b_2, it overlaps with a blockage. So we shift the macro b_2 upward to avoid the blockage and update the contour accordingly.

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 4 that the MP-tree leads to significantly better placement quality than that obtained by the traditional hierarchical method using independent packing trees.

To transform an MP-tree to its corresponding placement, each x-coordinate of a block can be determined by a DFS traversal, and the x-coordinate definition is the same as the packing tree. To compute 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 rectilinear region, respectively. Both BL- and BR-packing subtrees use the bottom-contour data structure, while the TL- and TR-packing subtrees use the top-contour one. 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 infeasible solutions.

3.3 Macro Placement Constraints

Based on the binary-tree representation, MP-tree can easily handle various placement constraints.

3.3.1 Placement Blockages and Pre-placed Macros

The placement blockages are given by the user, 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 y-coordinate of the macro to the position with no overlap. In the example shown in Figure 6, after adding a new macro b_2, it overlaps with the given placement blockage. We thus shift the macro b_2 upward to avoid the overlap, and update the contour accordingly. A pre-placed macro can be treated as a placement blockage, and there is no need to add such a corresponding node to the MP-tree.

3.3.2 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.

3.3.3 Rectilinear Macros

We adopt the method proposed in [15] to handle rectilinear macros for our MP-tree. A rectilinear macro is sliced into several rectangular blocks. The *location constraint* (*LC* for short) corresponding to the tree topology is created. During packing, we shift a *mis-alignemnt* macro upward to maintain the rectilinear block shape.

3.3.4 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 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, 1×4, 2×2, and 4×1. The desired clustering dimension is selected during simulated annealing, which will be explained later. For some macros, the timing between them is critical, we may also cluster these macros to satisfy the performance constraint.

3.3.5 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 sub-trees for its four corners so that macros can be packed along the region boundary.

3.4 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:

- Op1: Rotate a block or a cluster.
- Op2: Resize a cluster.
- Op3: Move a node in a packing subtree to another place.
- Op4: Swap two nodes within one or two packing subtrees.
- 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. For Op3, we select a node from a packing subtree, and move it to another place of the same or different packing subtrees. For Op4, we select two nodes from one (two) packing subtree(s), and swap them. For Op5, we swap two packing subtrees, and the packing order of two packing subtrees are exchanged. Note that the branch structure of an MP-tree is fixed and does not change by any type of operation.

3.5 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:

$$\Phi(F) = \alpha A + \beta W + \gamma D + \delta O,$$

where A is the *macro placement area*, W is the total *wirelength*, 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, wirelength, macro displacement, and vertical overlap length are explained in the following.

The macro placement area is the area under the bottom contour plus the area above the top contour. As shown in Figure 7(a), the contours are plotted by dashed lines, and the corresponding macro placement area is shown in Figure 7(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 wirelength will be smaller.

To consider the design hierarchy, we create pseudo nets between macros in the same design hierarchy group, based on the star model or the clique model [13]. By minimizing the total wirelength, the macros in the same design hierarchy group can be placed closer to each other.

Cost D is the total macro displacement, which is defined by

$$D = \sum_{blocks} (|x'_i - x_i| + |y'_i - y_i|)^2,$$

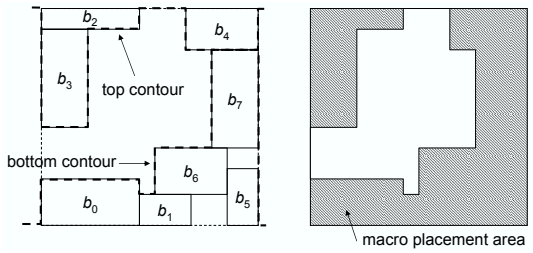


Figure 7: (a) A macro placement solution and its top and bottom contours. (b) The corresponding macro placement area.

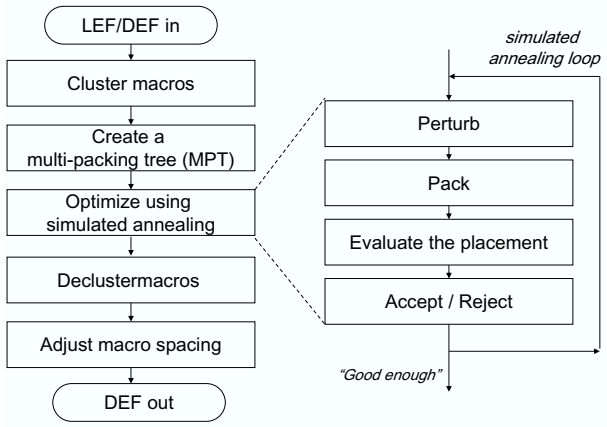


Figure 8: Our MP-tree macro placement flow. Note that this flow is the second step, macro placement, of the flow in Figure 1.

where (x_i, y_i) is the initial position of macro b_i, and (x'_i, y'_i) is the current position of macro b_i during 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 the bottom packing subtrees. However, there may exist vertical overlaps between the top contour and the bottom contour. The penalty cost O for the vertical overlap can guide the simulated annealing to find a non-overlap solution.

3.6 Macro Placement Flow

Figure 8 shows the flow for our MP-tree macro placer. After reading LEF/DEF files, we cluster the macros for the designated performance macros using the method discussed in Section 3.3.4, and the cluster dimension is initialized with the one closest to square since the square dimension 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 the region constraints are given, we need to create four subtrees for each region. Each macro/cluster corresponds to a node in a packing subtree. If an initial macro placement is given, we can assign the initial packing subtree to which a node belongs according to the nearest corner for the macro. Otherwise, we randomly set the initial packing subtree to which a node belongs. Each packing subtree is then initialized as a complete binary tree.

Simulated annealing is used to find the desired macro placement. We perturb an MP-tree to get another MP-tree by the aforementioned operations described in Section 3.4. After perturbing, 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 or not 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.

Finally, we adjust the spacing between macros. If the demand

Table 1: The 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%

of the routing resource between two macros is higher than the original spacing, we add more space between the two macros; otherwise, we can reduce the original spacing to make the macro placement area smaller. We also orientate macros by horizontal/vertical flipping to make the pins closer to the chip center. Then, we fix all macros and report the final macro placement solution.

4. EXPERIMENTAL RESULTS

To show the effectiveness and robustness of MP-tree, we conducted four experiments on an Opteron 2.6GHz machine. We used three state-of-the-art publicly available academic mixed-size placers, NTUplace3 [8], Capo 10.2 [10], and mPL6 [4].

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 mPL6 and Capo 10.2 to evaluate the effectiveness of the MP-tree with different placers. For the last experiment, we used five real industry designs to show the superiority of the MP-tree on both half-perimeter wirelength (HPWL) and routability.

4.1 Effects of Chip Utilization Rates

In this experiment, we used the ISPD'06 Placement Contest Benchmarks [1]. We changed all fixed macros in the benchmarks to movable ones to test our macro placement algorithm. Table 1 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 MP-tree. The area of these macros are 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 2 shows the resulting HPWL's with different chip utilization rates. The columns "w/o" give the resulting HPWL's using NTUplace3 alone while the columns "MPT" give the resulting HPWL's using NTUplace3 integrated with the MP-tree.

Integrating the MP-tree with NTUplace3, we can obtain legal placements with shorter HPWL's for most circuits; in contrast, NTUplace3 alone may not obtain legal placements for several benchmark circuits. Especially, 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 HPWL's 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. Notice that the circuit newblue3 has the highest MA-ratio, 83%, and NTUplace3 alone could not find any legal placement for this circuit. In contrast, NTUplace3 with the MP-tree generated legal placements robustly under different chip utilizations.

4.2 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 3.2.

Table 2: The resulting HPWL's 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

Table 3: The resulting HPWL's of using the MP-tree and four packing trees for 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

The method of using four packing trees is a simple extension to macro placement for a rectangular chip. We divided a chip into four sub-regions and created four different BL-/BR-/TL-/TR-packing trees in the corresponding sub-regions. Notice that although this extension for packing trees still can handle 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 3. From the results, we observed that the MP-tree is more robust in finding legal placements for all benchmarks while the method with four packing trees cannot. For those benchmarks with 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.

4.3 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 4 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. 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 10.2 reduced the average HPWL by 12% than that without the MP-tree. We tried several times, but mPL6 alone could not obtain legal solutions for seven circuits. With the MP-tree, however, mPL6 can obtain legal solutions for all circuits. This shows that the MP-tree is robust in finding legal solutions.

4.4 Routing Results on mchip Circuits

In this experiment, we show that the macro placements generated by the MP-tree lead to not only shorter HPWL's but also better routability. We used five real industry circuits, which include cell phones, DVD players, PDAs, etc.; the statistics are listed in Table 5. The numbers of cells range from 540K to 1320K, and the numbers of macros range from 50 to 380. There are also some pre-placed blocks and some blockages in these circuits. From the previous experiments, we found that Capo 10.2 can find legal results for mixed-size placement with large macros, and can handle pre-placed blocks and blockages correctly. As mentioned in the classification of the mixed-size placement methods in Section 1.1, the first type might not generate overlap-free macro positions. Therefore, we compared the MP-tree based macro placement (the third type) with Capo's (the second type). That is,

Table 4: The resulting HPWL's 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 10.2				mPL6			
	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

Table 5: The 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%
mchip5	1230k	1260k	58%	138	30%

our MP-tree and Capo are used to place macros only, and the remaining cells are placed by Synopsys Astro. After placement, we performed the global routing.

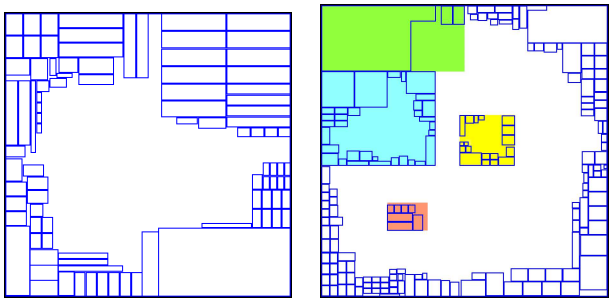
Table 6 shows the resulting HPWL's, routed wirelengths (WL), GRC overflows, and max overflows. The GRC overflow is the percentage of the global routing cells (GRC's) that have overflows. The larger the value, the more congested the placement. "Max overflow" gives the number of extra tracks assigned for the global routing cell with the maximum overflow.

For the five circuits, our MP-tree consistently obtains much better wirelengths (HPWL and WL) than Capo's macro placement. For the mchip5 circuit, segmentation faults occurred and could not be solved after several tries when using Capo. Further, Capo's macro placement results in larger GRC overflows and max overflows and needs more running time 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.

Figure 9(a) shows the macro placement result of mchip2 which contains 95 macros by using the MP-tree. Figure 9(b) shows the placement of mchip4 with 380 macros and four region constraints.

5. CONCLUSIONS

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 obtains much smaller wirelengths than leading academic mixed-size placers alone. Integrating our macro placer with the state-of-the-art academic standard-cell placers, such as Capo 10.2, mPL6, and NTUplace3, we can easily find legal



(a) mchip2 (b) mchip4

Figure 9: (a) The macro placement of mchip2 with 95 macros by using the MP-tree. (b) A macro placement of mchip4 with 380 macros and four region constraints.

Table 6: Comparison of Capo's macro placer and our MP-tree macro placer. NR: No placement result can be used for routing due to the segmentation fault in Capo.

	Capo's Macro Placement + Commercial Std-Cell Placer					
	Place		Route			
Circuit	HPWL (×e7)	Time (min)	WL (×e7)	Time (min)	GRC Overflow	Max Overflow
mchip1	5.84	16	6.56	23	0.7%	39
mchip2	5.65	28	6.65	32	1.0%	27
mchip3	10.00	23	16.90	180	36.4%	113
mchip4	14.12	41	14.16	323	1.4%	288
mchip5	Seg. fault		NR	NR	NR	NR
Comp.	1.35	1.73	1.55	6.83	97.25	13.02

	MP-tree + Commercial Std-Cell Placer					
	Place		Route			
Circuit	HPWL (×e7)	Time (min)	WL (×e7)	Time (min)	GRC Overflow	Max Overflow
mchip1	5.26	8	6.13	7	0.7%	5
mchip2	4.72	13	5.34	8	0.1%	4
mchip3	5.26	16	6.02	14	0.1%	4
mchip4	11.76	31	13.27	45	0.1%	31
mchip5	8.92	30	9.85	27	0.0%	2
Comp.	1.00	1.00	1.00	1.00	1.00	1.00

mixed-size placement results with significantly better wirelength and routability.

6. ACKNOWLEDGMENTS

This work was supported by MediaTek Inc. and partially supported by National Science Council of Taiwan under Grant No's NSC 95-2221-E-002-372, NSC 95-2221-E-002-374, NSC 95-2752-E-002-008-PAE.

7. REFERENCES

[1] *ISPD 2006 Placement Contest.* http://www.sigda.org/ispd2006/contest.html.
[2] S. N. Adya and I. L. Markov. Combinatorial techniques for mixed-size placement. *ACM Transactions on Design Automation of Electronics Systems*, 10(5), Jan. 2005.
[3] 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. Computer-Aided Design*, 24(5):748–761, May 2005.
[4] T. Chan, J. Cong, J. Shinnerl, K. Sze, and M. Xie. mPL6: Enhanced multilevel mixed-size placement. In *Proc. of ACM ISPD*, pages 212–214, San Jose, CA, Apr. 2006.
[5] C.-C. Chang, J. Cong, and X. Yuan. Multi-level placement for large-scale mixed-size ic designs. In *Proc. of IEEE/ACM ASP-DAC*, pages 325–330, Kitakyushu, Japan, 2003.
[6] Y.-C. Chang, Y.-W. Chang, G.-M. Wu, and S.-W. Wu. B*-trees: A new representation for non-slicing floorplans. In *Proc. of ACM/IEEE DAC*, pages 458–463, Los Angeles, CA, June 2000.
[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. of ACM ISPD*, pages 236–238, San Francisco, CA, Apr. 2005.
[8] 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. of IEEE/ACM ICCAD*, San Jose, CA, Nov. 2006.
[9] A. B. Kahng and Q. Wang. A faster implementation of APlace. In *Proc. of ACM ISPD*, pages 218–220, San Jose, CA, Apr. 2006.
[10] J. Roy, D. Papa, A. Ng, and I. Markov. Satisfying whitespace requirements in top-down placement. In *Proc. of ACM ISPD*, pages 206–208, San Jose, CA, Apr. 2006.
[11] P. Spindler and F. M. Johannes. Fast and robust quadratic placement combined with an exact linear net model. In *Proc. of IEEE/ACM ICCAD*, San Jose, CA, Nov. 2006.
[12] T. Taghavi, X. Yang, B.-K. Choi, M. Wang, and M. Sarrafzadeh. Dragon2006: Blockage-aware congestion-controlling mixed-size placer. In *Proc. of ACM ISPD*, pages 209–211, San Jose, CA, Apr. 2006.
[13] N. Viswanathan and C. C.-N. Chu. Fastplace: Efficient analytical placement using cell shifting, iterative local refinement and a hybrid net model. In *Proc. of ACM ISPD*, pages 26–33, Apr. 2004.
[14] E. Wein and J. Benkoski. Hard macros will revolutionize SoC design. *EETimes Online*, Aug. 2004. http://www.eetimes.com/showArticle.jhtml?articleID=26807055.
[15] G.-M. Wu, Y.-C. Chang, and Y.-W. Chang. Rectilinear block placement using B*-trees. *ACM Transactions on Design Automation of Electronics Systems*, 8(2):188–202, 2003.