IMF: Interconnect-Driven Multilevel Floorplanning for Large-Scale Building-Module Designs

Tung-Chieh Chen
Graduate Institute of Electronics Engineering
National Taiwan University
Taipei 106, Taiwan
Email: tungchieh@ntu.edu.tw

Yao-Wen Chang
Graduate Institute of Electronics Engineering and
Department of Electrical Engineering
National Taiwan University
Taipei 106, Taiwan
Email: ywchang@cc.ee.ntu.edu.tw

Shyh-Chang Lin
SpringSoft, Inc.
Hsin-Chu 300, Taiwan
Email: chris@springsoft.com.tw

***Abstract*— We present in this paper a new interconnect-driven multilevel floorplanning, called IMF, to handle large-scale building-module designs. Unlike the traditional multilevel framework that adopts the "V-cycle" framework: bottom-up coarsening followed by top-down uncoarsening, in contrast, IMF works in the "Λ-cycle" manner: top-down uncoarsening (partitioning) followed by bottom-up coarsening (merging). The top-down partitioning stage iteratively partitions the floorplan region based on min-cut bipartitioning with *exact net-weight modeling* to reduce the number of global interconnections and thus the total wirelength. Then, the bottom-up merging stage iteratively applies fixed-outline floorplanning using simulated annealing for all regions and merges two neighboring regions recursively. We also propose an accelerative fixed-outline floorplanning (AFF) to speed up wirelength minimization under the outline constraint. Experimental results show that IMF consistently obtains the best floorplanning results with the smallest wirelength for large-scale building-module designs, compared with all publicly available floorplanners. In particular, IMF scales very well as the circuit size increases.**

The Λ-cycle multilevel framework outperforms the V-cycle one in the optimization of global circuit effects, such as interconnection and crosstalk optimization, since the Λ-cycle framework considers the global configuration first and then processes down to local ones level by level and thus the global effects can be handled at earlier stages. The Λ-cycle multilevel framework is general and thus can be readily applied to other problems.

I. Introduction

As nanometer IC technologies advance, design complexity is growing at a dramatic speed. Modern chip designs often consist of millions of transistors and designs with billions of transistors are already in production. To cope with the increasing design complexity, IP modules are widely reused for large-scale designs. Therefore, efficient and effective design methodology and tools capable of placing and optimizing large-scale modules are essential for modern chip designs.

A. Framework Evolution

The floorplanning frameworks are evolving to tackle the challenges with constantly increasing design complexity. Three major frameworks have been extensively studied in the literature: the flat, hierarchical, and multilevel frameworks. Many flat algorithms based on various floorplan representations have been proposed in the literature [14], [20], [23], [29]–[31], [33], [34], [37]. However, these algorithms does not scale well as the design size increases. To cope with the scalability problem, hierarchical approaches are proposed. The hierarchical approaches recursively divide a floorplanning region into a set of sub-regions and solve those sub-problems independently. Adya et al. [9] propose a "floorplacement" framework (used in their program Capo 9) that combines partitioning and floorplanning techniques to handle both floorplanning and placement problems. It first partitions a floorplan and then finds legal sub-floorplans. Cong et al. [18] present a fast floorplanner called PATOMA using look-ahead enabled recursive bipartitioning. It partitions a floorplan and uses row-oriented block (ROB) packing and zero-dead space (ZDS) floorplanning to find legal sub-floorplans. Both the floorplacement and PATOMA are based on the hierarchical framework in which the floorplanning stage is only used for legalization and overlap removal. The top-down hierarchical technique is efficient in handling large-scale problems. Nevertheless, a drawback of the hierarchical approaches is that they might lack the global information for the floorplanning interaction among different sub-regions.

To remedy the deficiency, the multilevel framework is proposed to solve the floorplanning problems (e.g., MB*-tree [28] and MLGFA [24]) as well as graph/circuit partitioning (e.g., Chaco [21], hMetis [27], ML [10]), placement (e.g., mPL [12]), and routing (e.g., MRS [17], MR [32], MARS [19], CMR [22]). All of the existing multilevel frameworks adopt a two-stage technique, bottom-up coarsening followed by top-down uncoarsening, which is known as the "V-cycle" framework. Lee et al. [28] first proposed a V-cycle multilevel floorplanning algorithm based on the B*-tree representation [14], called MB*-tree. It adopts a two-stage technique, clustering followed by declustering, based on the cost metric of area and connectivity. Hu et al. [24] proposed a V-cycle multilevel genetic floorplanning algorithm, called MLGFA. However, the algorithm only optimizes the chip area without considering the wirelength. Further, both of the multilevel floorplanning algorithms consider only variable-die floorplanning.

As pointed out by Kahng in [25], modern VLSI design is based on a fixed-die (fixed-outline) floorplan, rather than a variable-die one. A floorplan with pure area minimization without any fixed-outline constraint may be completely useless because it cannot fit into the given outline. Unlike classical floorplanning that usually handles only module packing to minimize silicon area, modern floorplanning should be formulated as a fixed-outline floorplanning with wirelength (interconnection) minimization. Since the chip area is given, we should optimize the wirelength and timing for fixed-die floorplanning to facilitate routing. Very few existing floorplanners can handle the fixed-die constraint with wirelength optimization. Among them, the floorplacer Capo 9 [9] and Parquet [6]–[8] (based on the sequence pair representation [33]) are probably the most popular fixed-die floorplanners with wirelength optimization mode. (Note that although most existing standard-cell/mixed-size placers can handle the large-scale circuit placement problem. They usually focus more on the standard-cells of the same heights. Therefore, the standard-cell/mixed-size placers cannot handle large-scale building-module floorplanning well. We have tried well-known publicly available placers such as Feng Shui 2.6/5.0 [1] and mGP [4], [13]. They all cannot obtain feasible or desirable building-module floorplans.)

The existing V-cycle multilevel framework handles the target problems first bottom-up from local configurations to global ones and then refines the solutions top-down from global to local. It is obvious that there are significant limitations for the V-cycle framework to handle the global circuit effect, such as interconnection and crosstalk optimization, since only local information is available at the beginning stages. A wrong choice made in such early stages may make the solution very hard to be refined during the top-down stage.

B. Our Contributions

In this paper, we present the first "Λ-cycle" (pronounced as the "lambda" cycle) multilevel framework. Unlike the traditional V-cycle multilevel frameworks that apply the bottom-up coarsening followed by the top-down un-coarsening, our Λ-cycle multilevel framework adopts the two-stage technique of top-down uncoarsening followed by bottom-up coarsening. The Λ-cycle multilevel framework outperforms the V-cycle one in the optimization of global circuit effects, since the Λ-cycle framework first considers the global configuration and then processes

down to local ones level by level and thus the global effects can be handled at earlier stages.

Based on the new framework, we develop the first Λ-cycle multilevel floorplanning algorithm to handle the interconnect-driven, large-scale floorplan designs. (It shall be noted that the Λ-cycle multilevel framework is general and thus can be readily applied to other problems as well.) Our Λ-cycle interconnect-driven multilevel floorplanning framework (IMF for short) adopts the two-stage technique, top-down partitioning followed by bottom-up merging. The top-down partitioning stage iteratively partitions the floorplan region based on min-cut bipartitioning with exact net-weight modeling to reduce the number of global interconnections and thus the total wirelength. Then, the bottom-up merging stage iteratively applies fixed-outline floorplanning using simulated annealing for all regions and merges two neighboring regions recursively. We also propose an accelerative fixed-outline floorplanning (AFF) to speed up wirelength minimization under the outline constraint. Experimental results show that IMF consistently obtains the best fixed-outline floorplanning results with the smallest wirelength for large-scale building-module designs, compared with all publicly available floorplanners B*-tree, MB*-tree, Parquet 3.1/4.0, and Capo 9. In particular, IMF scales very well as the circuit size increases.

The remainder of this paper is organized as follows. Section II compares our Λ-cycle multilevel framework with the V-cycle and the hierarchical ones. Section III presents our floorplanning algorithms. Section IV shows the experimental results, and finally the conclusions are given in Section V.

II. Multilevel Framework

The traditional V-cycle multilevel frameworks apply a two-stage technique, bottom-up coarsening followed by top-down uncoarsening. We take MB*-tree [28] for an example. Figure 1(a) shows the MB*-tree multilevel framework. It is based on a two-stage technique of bottom-up clustering followed by top-down declustering. The clustering stage iteratively groups a set of modules based on a cost metric of area and module connectivity. The declustering stage iteratively ungroups a set of the previously clustered modules and uses simulated annealing to refine the solution. Experimental results showed that MB*-tree obtains solutions of very small dead space. For modern floorplanning, however, the interconnections among modules are a very important cost metric for routability and performance optimization and thus should be carefully considered. Since MB*-tree first works in a bottom-up manner by clustering local modules based on area and local connectivity, it does not have the view for the global configuration at the earlier stages. Therefore, it is very likely that MB*-tree can only obtain a sub-optimal solution since it may make a wrong choice during the clustering stage, and it may become very hard to further refine the floorplan solution during the declustering stage.

Figure 1(b) illustrates our Λ-cycle interconnect-driven multilevel floorplanning framework (IMF). Unlike MB*-tree that adopts a V-cycle framework, our IMF uses the Λ-cycle of top-down partitioning followed by bottom-up merging (refinement). Section III presents our IMF algorithm.

Table I lists the characteristics of our IMF multilevel framework, the MB*-tree multilevel framework [28], the Capo floorplacemnet framework [9], and the PATOMA framework [18]. Our IMF framework and the MB*-tree framework are based on the multilevel framework while the Capo framework and the PATOMA framework are based on the hierarchical framework. Although Capo and PATOMA use partitioning, unlike IMF, they do not have the refinement stage to further improve their results.

III. Algorithm

The IMF algorithm consists of three steps: (1) chip dimension determination, (2) the partitioning stage, and (3) the merging stage.

A. Chip Dimension Determination

Given a set of modules with the total area A and the *maximum white-space fraction* γ, we can construct a fixed outline with the aspect ratio (height/width) α. The chip dimension (H^*, W^*) can be computed by the following equations [8]:

$$H^* = \sqrt{(1+\gamma)A\alpha}, \qquad W^* = \sqrt{(1+\gamma)A/\alpha}. \tag{1}$$

If $\max(H_i, W_i)$ of a module m_i is larger than $\max(H^*, W^*)$, the module m_i can never fit into the chip boundary. In this case, the chip dimension can be computed by the following equations:

$$H^* = \max(W_i, H_i), \qquad W^* = \frac{(1+\gamma)A}{H^*}. \tag{2}$$

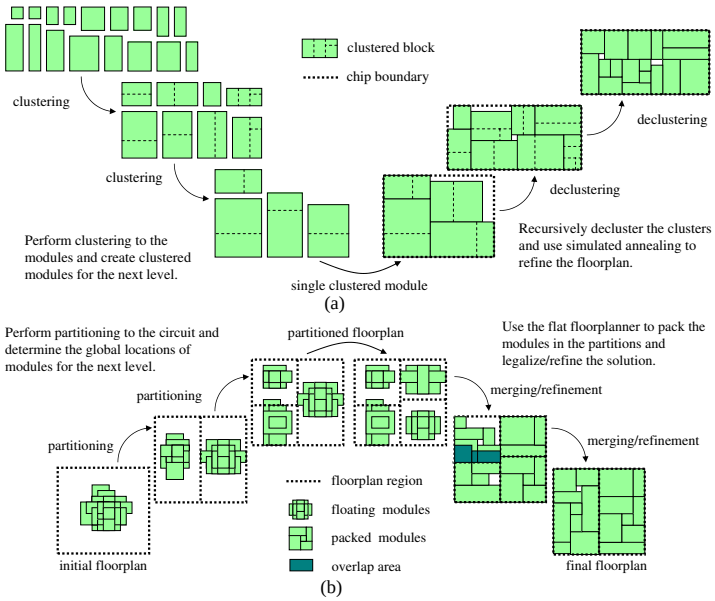


Fig. 1. (a) The V-cycle multilevel framework of MB*-tree. (b) The Λ-cycle multilevel framework of IMF.

TABLE I
Framework comparisons.

Framework	Characteristics
Our IMF multilevel framework	• Use the Λ-cycle multilevel framework. • Use top-down partitioning followed by bottom-up merging. • Handle fixed-die constraints. • Minimize the wirelength under the given area constraint.
The MB*-tree multilevel framework in [28]	• Use the V-cycle multilevel framework. • Use bottom-up clustering followed by top-down declustering. • Deal with variable dies and cannot guarantee to satisfy an outline constraint. • Need to specify the weights for area and wirelength by the user.
The Capo 9 floorplacement framework in [9]	• Use the top-down hierarchical framework. • Use partitioning and fixed-outline floorplanning. • Minimize the wirelength under the given chip-outline. • Do not have a refinement stage.
The PATOMA framework in [18]	• Use the top-down hierarchical framework. • Use partitioning and ZDS/ROB fast look-ahead floorplanning. • Minimize the wirelength under the given chip-outline. • Do not have a refinement stage.

The new dimension of the chip can ensure that every module fit into the chip boundary. Note that the chip area $A^* = H^*W^* = (1+\gamma)A$ remains the same as the original formulation.

B. The Partitioning Stage

At the initial level, the locations of all modules are set to the center of the chip region. To prevent from generating sub-regions of large aspect ratios, we choose the longer side to divide the region into two sub-regions. After the shapes of two sub-regions are determined, we move the modules to the two centers of the two sub-regions to minimize the half-perimeter wirelength (HPWL).

The module-location determination problem can be formulated as a hypergraph partitioning problem. We first derive an exact net-weight modeling to map the HPWL cost exactly to the min-cut cost. With the exact modeling, in other words, minimizing HPWL is equivalent to finding the min-cut cost. Therefore, the given hypergraph is partitioned using a min-cut bipartitioner to obtain the minimum HPWL. The new locations of the modules are thus determined by the partitioner, and each sub-partition corresponds to a sub-region.

1) Exact Net-Weight Modeling: Since the net weight in the traditional terminal propagation for the min-cut based placement is a constant value, the weight with the change in HPWL cannot be exactly modelled, whether a net is cut or not. The underlying idea for our exact net-weight modeling (terminal propagation) is that we want to map the min-cut cost exactly to the HPWL change, which is similar to the "bounding-box aware terminal propagation (BBTP)" proposed by Selvakkumaran and Karypis

in [35], [36]. They first proposed BBTP in [35] and later discussed the BBTP in detail under seven cases in [36]. Another net-weighting method was proposed in [16], they discussed the net-weighting method for partitioning based on four cases. However, they can obtain exact modeling only for two-terminal nets, i.e., they can only obtain suboptimal results for multi-terminal nets. Unlike the previous work that exhaustively enumerates of potential cases, we derive a unified model to assign the net weights to map the HPWL value exactly. Our HPWL modeling not only can be applied to vertical-cut or horizontal-cut partitioning, but can also be applied to placement feedback (repartitioning) [26]. Further, our unified HPWL model can even apply to the partitioning associated with two non-adjacent regions, for which the method presented in [36] would need to enumerate tens of cases for the HPWL modeling and thus is obviously much more complex and harder for implementation.

We give our unified HPWL modeling as follows. A circuit is modelled as a hypergraph. Each node in the hypergraph corresponds to a module inside the target region, with the node weight being set to the area of the corresponding module. Each hyperedge denotes a multi-terminal net in the circuit, with the hyperedge weight being set to the value of the HPWL contribution if the hyperedge is cut.

The hyperedge weight can be determined as follows. For easier presentation, we take Figure 2 as an example to explain the unified HPWL (net-weight) modeling. The discussion readily applies to other cases. Consider a 4-terminal net with two fixed terminals and two modules. The x-*range* of the two terminals (i.e., the span between the two terminals in the x direction) is within that of the centers of the two partitions, and the center of the left partition is closer to the x-range. For each net, we compute three HPWL values. Let w_1 be the HPWL when the two modules are both at the side closer to the span of the terminals. In Figure 2(a), the two modules are at the left side. w_1 is equal to the half of the bounding box shown in Figure 2(a), represented by the dotted lines. Let w_2 be the HPWL when the two modules are at the *opposite* side (the right side for the example shown in Figure 2(b)). Similarly, w_{12} is the HPWL when the two modules are at different sides (Figure 2(c)). For the case shown in Figure 2, it is easy to see that $w_{12} > w_2 > w_1$.

For each case shown in Figure 2, we introduce a partitioning hypergraph with two fixed nodes to represent the two sides and two movable nodes to represent the two movable modules. We then add two hyperedges e_1 and e_2 into the hypergraph. For the case of Figure 2 where the center of the left partition is closer to the x-range, we introduce e_1 to connect the left fixed node and the two movable nodes and e_2 to connect between the two movable nodes. We then assign the weight of the hyperedge e_1 with the value $w_2 - w_1$ (here, $w_2 > w_1$), and that of the hyperedge e_2 with the value $w_{12} - w_2$. Partitioning the resulting hypergraph can determine to which partition the module belongs. There are three possible partitioning results, as shown in the Figures 2(d), (e), and (f). The three partitioning results correspond to the configurations shown in Figures 2(a), (b), and (c), respectively. For the case of Figure 2(d), no hyperedge in the resulting hypergraph is cut. Therefore, its cutsize $n_{cut} = 0$. In Figure 2(e), e_1 is cut, and the cutsize is given by $n_{cut} = w_2 - w_1$. In Figure 2(f), both e_1 and e_2 are cut, and thus the cutsize $n_{cut} = w_{12} - w_1$. For all of these three cases, we conclude that the corresponding HPWL is given by $w_1 + n_{cut}$. In particular, the theoretical result holds for the general cases with 2- or multi-terminal nets, and we have the following theorem:

Theorem 1: With the unified net-weight modeling, we have $HPWL = w_1 + n_{cut}$.

Note that Theorem 1 holds for other cases though not discussed here. Let $w_{1,i}$ be the HPWL of net i when its modules are all at the side closer to the span of the terminals (see Figure 2(a)) and $n_{cut,i}$ be the cutsize of net i. By Theorem 1, we have

$$\begin{aligned} \min(\sum HPWL_i) &= \min(\sum (w_{1,i} + n_{cut,i})) \\ &= \sum w_{1,i} + \min(\sum (n_{cut,i})), \end{aligned}$$

since $\sum w_{1,i}$ is a constant. Thus, finding the minimum HPWL is equivalent to finding the min-cut as long as the external terminals are given.

Theorem 2: The unified net-weight modeling exactly maps HPWL to the min-cut cost.

The partitioning stage continues until the number of modules in each partition is smaller than a threshold. Then, the partitioned floorplan is obtained.

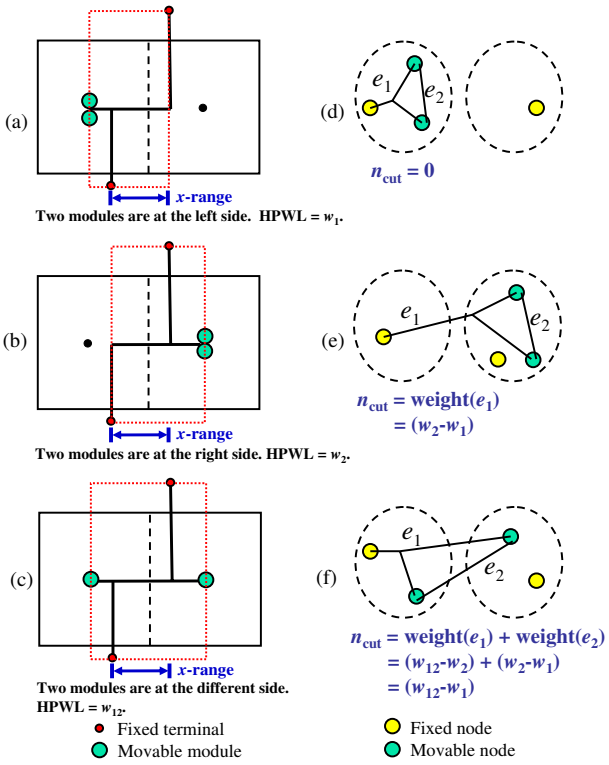
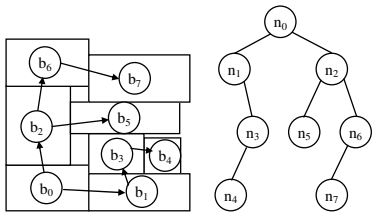


Fig. 2. An example of determining a net weight. (a), (b), and (c) are three possible partitioning results. (d), (e), and (f) are corresponding partitioning hypergraphs.



(a) (b)

Fig. 3. (a) An admissible placement. (b) The B*-tree representing the placement.

C. The Merging Stage

In the merging stage, we first use fixed-outline floorplanning to pack the modules in the partition, and then merge two neighboring regions into one larger region. The fixed-outline floorplanning is applied again to legalize/refine the floorplan.

1) Fixed-Outline Floorplanning: Each region has its own height and width, and all modules in the region must fit into the region to generate a feasible floorplan. We treat the modules and I/O pads outside the current region as fixed terminals. We use the B*-tree representation with simulated annealing to find a feasible floorplan within the region. The reasons are two-fold: (1) The B*-tree has been shown an efficient and effective data structure for floorplan design [14], and (2) we intend to make fair comparison with the state-of-the-art multilevel floorplanning work MB*-tree [28], which is also based on the B*-tree.

We shall review the B*-tree floorplan representation [14]. Given an admissible placement in which all modules cannot be move to the left or to the bottom [20], we can construct a unique B*-tree in linear time to model the placement. Figure 3 shows an admissible placement and its corresponding B*-tree. A B*-tree is an ordered binary tree whose root corresponds to the module on the bottom-left corner. Similar to the depth-first search (DFS) procedure, we construct a B*-tree T for an admissible placement in a recursive fashion: Starting from the root, we first recursively construct the left subtree and then the right subtree. Let R_i denotes the set of modules located on the right-hand side and adjacent to b_i. The left child of the node n_i corresponds to the lowest module in R_i that is unvisited. The right child of n_i represents the lowest module located above and with its x-coordinate equal to that of b_i. Given

a B*-tree, the x-coordinates of all modules can be easily determined by traversing the tree once [14], and we can apply a contour structure [20] to compute the y-coordinates in amortized linear time.

The cost function Φ for simulated annealing is similar to the one in [15], and it is defined as follows:

$$\Phi = k_1 \frac{A_F}{A_{F,norm}} + k_2 \frac{W_L}{W_{L,norm}} + k_3 \left(\frac{W_F}{H_F} - \frac{W_R}{H_R} \right)^2, \qquad (3)$$

where A_F is the current floorplan area, $A_{F,norm}$ is the area normalization factor, W_L is the current wirelength, $W_{L,norm}$ is the wirelength normalization factor, W_F is the current floorplan width, H_F is the current floorplan height, W_R is the width of the region, H_R is the height of the region, and k_1, k_2, k_3 are user-specified parameters. To calculate the area/wirelength normalization factors, several times of random perturbations are performed before simulated annealing starts, and $A_{F,norm}$ ($W_{L,norm}$) is set to the average value of A_F (W_L).

2) Accelerative Fixed-Outline Floorplanning: We observe that it takes much more time for fixed-outline floorplanning than for partitioning/merging. Therefore, we propose an *accelerative fixed-outline floorplanning* technique (*AFF* for short) to speed up the whole framework. Typically, fixed-outline floorplanning spends most time in computing the wirelength, as also observed in [11]. Further, many floorplanning results might not be feasible because the resulting floorplans cannot fit into the bounding box. To speed up floorplanning, we first set $k_2 = 0$ for the cost function to perform area-driven fixed-outline floorplanning. Then, *we calculate the wirelength only when the floorplan can fit into the bounding box with a smaller cost*. If the resulting wirelength is better than the best wirelength, we save the current result as the best result. Since we can reduce significant running time for computing the wirelength, we may increase the number of perturbations to search for better floorplan solutions. It will be clear later that the method can reduce significant running time in wirelength computation without trading too much solution quality, especially for large-scale circuits (see Section IV-B).

3) Partition Merging: If the fixed-outline floorplanning cannot find a feasible floorplan within the bounding box, we still keep the solution. In the next refinement level, two partitioned regions are merged. To merge two vertical regions, we make the root of the B*-tree for the upper sub-floorplan as the right child of the right-most node of the B*-tree for the bottom sub-floorplan. The width of the merged floorplan is equal to the maximum width of the sub-floorplans, and the height of the floorplan is less than or equal to the sum of the two sub-floorplan's heights due to the packing. To merge two horizontal regions, we first find the node which corresponds to the right-most module of the left sub-floorplan. Then, we make the root of the B*-tree for the right sub-floorplan as the left child of the node we found. The height of the merged floorplan is equal to the maximum height of the two sub-floorplans, and the width of the merged floorplan is equal to the sum of the two sub-floorplan's widths.

The merging stage iteratively merges two previously partitioned regions and then refines the floorplan solution based on fixed-outline simulated annealing. The merging stage continues until all regions are merged into one top-most region, and the final floorplan is obtained.

D. Algorithm

Figure 4 summarizes our algorithm. The inputs are dimensions of modules, a (multi-terminal) netlist, the location of the I/O pads, and fixed-die parameters. We first initialize our data structures and determine the chip boundary. Then, all modules are set to the center of the floorplaning region. In the partitioning stage, we create a hypergraph for the current region and apply a state-of-the-art hypergraph/circuit partitioner, such as hMetis [3], to obtain a min-cut bipartitioning result. (By using hMetis which is based on the V-cycle multilevel framework, our implementation applies the Λ-cycle in the whole framework with V-cycles in the partitioning stage.) The modules are then moved to the centers of the sub-regions according to the partitioning result. The partitioning stage continues until every region has fewer than n_{max} modules, and the partitioned floorplan is obtained. In the merging stage, the fixed-outline floorplanning with wirelength minimization is applied to pack the modules into the regions. Then, two regions are merged into a larger one. After all regions are merged, we obtain the final floorplan.

IV. Experimental Results

We made the comparisons with the following five state-of-the-art floorplanning algorithms/packages: our IMF, B*-tree [14], Parquet-3.1/-4.0 [5], MB*-tree [28], and Capo 9 [9] (version 9.0r4). We used the MCNC and the GSRC [2] benchmark suites. All programs were compiled with gcc 3.3.2, and all experiments were preformed on a Linux PC with an Intel Pentium 4 3.2GHz CPU with 3 GB memory. We convert the benchmarks from the Bookshelf floorplanning format to the Bookshelf placement format so that Capo can run on them. (We ran Capo in the default mode. By default, Capo has 3 iterations of placement feedback [26], which leads to better partitioning results. Our current IMF, however, does not have placement feedback, and we believe that our results can be further improved if the placement feedback is applied.) Note that the PATOMA floorplanner [18] mentioned in Section I is not available to the public and the poster paper [18] does not report any results, so we are not able to compare with PATOMA. (We have also tested publicly available mixed-size placers on the floorplan benchmarks, including Feng Shui 2.6/5.0 [1] and mGP [4], [13]. Feng Shui generated the floorplanning results directly using its legalizer without performing global placement. Thus, its results are far from optimal. For mGP, it results in many overlaps and places some modules outside the chip boundary. So we shall not compare with those mixed-size placers.)

For fair comparisons, we first set the maximum white-space fraction γ to 15%, and chip aspect ratio α to 1 for all circuits. The I/O pads were scaled to the chip boundary. The wirelength was estimated using half-perimeter wirelength (HPWL).

Interconnect-Driven Multilevel Floorplanning (IMF)	
Input: Modules, nets, I/O pads, fixed-die parameters. Output: A feasible floorplan within the fixed-die with wirelength (HPWL) being minimized.	
1.	Initialize the data structures and the chip dimension;
2.	Set all modules at the center of the floorplan region;
3.	**The Partitioning Stage**:
4.	**until** every region has fewer than n_{max} modules
5.	Choose a partition;
6.	Create a hypergraph model;
7.	Bipartition the hypergraph;
8.	Move modules to the new sub-regions;
9.	Generate the partitioned floorplan;
10.	**The Merging Stage**:
11.	**while** there exists more than one region
12.	Choose two neighboring regions;
13.	Apply fixed-outline floorplanning;
14.	Merge two regions;
15.	Generate the final floorplan;
16.	**return** the final floorplan;

Fig. 4. The IMF algorithm.

TABLE II
COMPARISONS OF THE HPWL BY USING THE TRADITIONAL TERMINAL PROPAGATION (TTP) AND THE EXACT NET-WEIGHT MODELING (ENW). IN EACH ENTRY, BOTH THE **minimum/average** VALUES OBTAINED IN TEN RUNS ARE REPORTED. THE TTP/ENW RATIO IS THE RATIO OF THE TTP RESULT AND THE ENW RESULT.

Circuit	TTP	ENW	TTP/ENW
n100	202713/203685	191199/191382	1.06/1.07
n200	365047/366104	341425/341660	1.07/1.07
n300	478947/479934	450806/453806	1.06/1.06
Average	–	–	1.06/1.07

A. Exact Net-Weight Modeling

Table II compares the net-weight modeling based on the traditional terminal propagation (TTP for short) and our exact net-weight modeling (ENW for short). For TTP, all net-weights are equal to 1.0. For ENW, the weights of the nets are assigned according to the aforementioned scheme described in Section III-B.1. The experiments were taken on the three largest GSRC benchmark since they are better for testing the effectiveness of the partitioning. We used the state-of-the-art V-cycle multilevel partitioner hMetis [3] (version 1.5.3) for min-cut partitioning. We performed partitioning on each circuit ten times to get the minimum/average HPWL. We set $n_{max} = 10$ (i.e., partition all regions until each of them contains fewer than 10 modules). From Table II, we observe that the exact net-weight modeling can reduce the HPWL by 6%–7%. Note that the runtimes are not reported here for both methods since they are about the same.

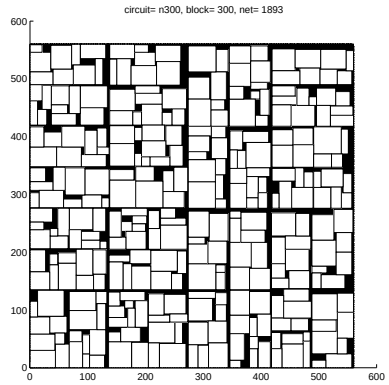


Fig. 5. The resulting floorplan for the circuit n300.

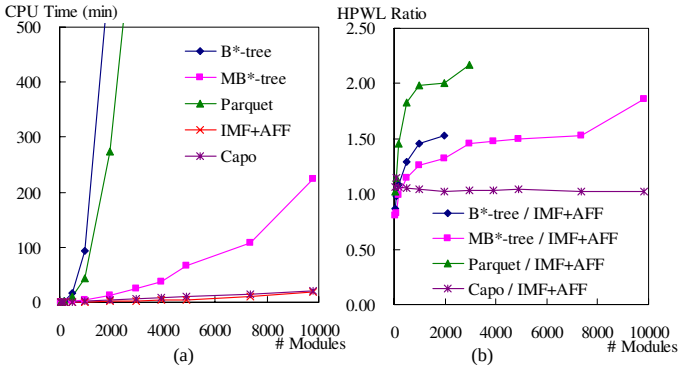


Fig. 6. (a) Comparison for the CPU time vs. circuit size (# of modules). (b) Comparison for the HPWL ratio vs. circuit size (# of modules).

B. Comparisons of Solution Quality

Table III lists the HPWL's, dead spaces, and CPU times obtained by B*-tree, MB*-tree, Parquet, IMF, IMF with accelerative fixed-outline floorplanning (IMF+AFF), and Capo for the MCNC/GSRC benchmarks. The number of modules ranges from 9 to 300, and the number of nets ranges from 83 to 1893. The average HPWL ratio is normalized to the result of IMF. The total time gives the summation of the CPU times for all circuits. Since B*-tree and MB*-tree cannot handle fixed-die constraints, we set their mode to *wirelength optimization alone* to make comparisons. By doing so, B*-tree and MB*-tree should gain some advantages. In contrast, Parquet is a fixed-outline floorplanner (which is based on the sequence pair (SP) representation). (Note that Parquet can also use the B*-tree representation. Nevertheless, we still compare with the SP representation since Parquet obtains better results with the SP representation mode [11]. For fair comparison, the I/O pads for all circuits are fixed along the user-specified chip boundaries. Thus, the I/O pad locations for variable-die floorplanners are the same as those for fixed-die floorplanners. Further, we reported the best results obtained by Parquet 3.1 and 4.0 since neither version dominated the other.) We used the default wirelength minimization mode (`-minWL`), and set the maximum white-space fraction $\gamma = 0.15$ (`-maxWS 15`) and chip aspect ratio 1.0 (`-ar 1.0`), except that the aspect ratio of the circuit `hp` was set to 1.075 to allow all modules to fit into its bounding box. The average dead space of the B*-tree (MB*-tree) variable-die floorplanner is about 24.1% (26.2%), and none of its resulting floorplans fits into the bounding box. The dead spaces of the fixed-die floorplanners are all less than 15.0%, and all results can fit into the bounding boxes. (However, the floorplacer Capo does not legalize all modules for the benchmark circuit `hp`.)

As shown in Table III, IMF reduces the HPWL by 10% on average compared to Parquet. Although B*-tree (MB*-tree) minimizes the HPWL alone, it only reduces the HPWL by 4% (1%) and incurs significantly larger dead spaces compared to IMF. The CPU times for B*-tree and Parquet are comparable, while the multilevel floorplanners, MB*-tree and IMF, and the hierarchical floorplacer Capo, spend much less CPU time, especially for larger circuits. IMF+AFF achieves 11X speedup at the cost of 9% HPWL overhead compared to IMF. Based on the results, AFF can significantly reduce the running time, and it is particularly suitable for large-scale circuits since it achieves larger speedup (e.g., 16.7X for `n300`) with only 1% overhead in the HPWL value. So the AFF option is turned on when handling large-scale circuits. The runtime of Capo floorplacer is between that of IMF and IMF+AFF. The solution quality of Capo is 11% and 2% worse than IMF and IMF+AFF respectively. Note that for the circuit `hp`, we have tried several runs, but Capo always reports overlaps in the floorplan results. Figure 5 shows the resulting floorplan of `n300` using IMF+AFF.

C. Scalability of the Floorplanners

To test the scalability of the five algorithms/floorplanners, we tested on the large-scale floorplan benchmark circuits used in [28], which are duplicated from the largest MCNC benchmark circuit `ami49` to generate larger test circuits. For [28], they simply duplicated all modules and nets of the circuit `ami49`. However, these kinds of synthetic circuits are not general since there is no interconnection between the duplicated copies of circuits. The clustering (partitioning) method would take advantage of these kinds of special structures. To make the comparison more fair, we also added interconnections among different copies of duplicated circuits. For the circuit `ami49_x`, we duplicated each module/net x times. For each module m_i, we duplicated it as $m_{i,1}, m_{i,2}, ..., m_{i,x}$ and added $x-1$ nets between $(m_{i,1}, m_{i,2})$, $(m_{i,1}, m_{i,3})$, ..., $(m_{i,1}, m_{i,x})$. We divide block widths/heights by 5 for the benchmarks to avoid overflows in computing the wirelength for MB*-tree.

Table IV lists the results for the five algorithms/floorplanners on `ami49_x` circuits. The average HPWL ratio is normalized to the HPWL of IMF+AFF. In Figures 6(a) and (b), the resulting CPU times and HPWL for the five algorithms are plotted as functions of the circuit size (in the number of modules), respectively. Figure 6(a) reveals that the CPU times for B*-tree and Parquet grow dramatically as the circuit size increases while both IMF+AFF and Capo can scale to very large-scale designs. As shown in Table IV and Figure 6(b), our IMF+AFF consistently obtains the best HPWL with *feasible* floorplans and on average outperforms B*-tree, MB*-tree, Parquet, and Capo by about 20%, 29%, 56%, and 5%, respectively. The resulting HPWL ratios for Parquet and MB*-tree grow up to 50% or more as the circuit size increases. The experimental results show that IMF+AFF has superior scalability and maintains high-quality results for large-scale designs.

V. Conclusion

We have presented a new interconnect-driven multilevel floorplanning algorithm (IMF) based on the novel Λ-cycle multilevel framework. IMF adopts a two-stage technique, partitioning followed by merging. The exact net-weight modeling is used in the partitioning stage, and an accelerative fixed-outline floorplanning (AFF) is applied in the merging stage. Experimental results show that IMF+AFF scales very well as the circuit size and interconnection complexity increase. For large-scale floorplan designs, IMF outperforms all state-of-the-art floorplanners that are available to the public in wirelength optimization, including B*-tree, MB*-tree, and Parquet (and the floorplacer Capo). The Λ-cycle multilevel framework proposed in this paper is general and thus can be applied to other problems.

VI. Acknowledgments

This work was partially supported by SpringSoft, Inc. and National Science Council of Taiwan under Grant No's. NSC 93-2215-E-002-009, NSC 93-2220-E-002-001, and NSC 93-2752-E-002-008-PAE.

References

[1] *FengShui Placer*. http://vlsicad.cs.binghamton.edu/software.html.
[2] *GSRC Floorplan Benchmarks*. http://www.cse.ucsc.edu/research/surf/GSRC/progress.html.
[3] *hMetis*. http://www-users.cs.umn.edu/∼karypis/metis/hmetis/.
[4] *mGP: Multilevel Global Placement*. http://ballade.cs.ucla.edu/mGP/.
[5] *PARQUET*. http://vlsicad.eecs.umich.edu/BK/parquet/.
[6] S. Adya and I. Markov. Fixed-outline floorplanning through better local search. In *Proc. of ICCD*, pages 328–333, 2001.
[7] S. Adya and I. Markov. Consistent placement of macro-blocks using floorplanning and standard-cell placement. In *Proc. of ISPD*, pages 12–17, 2002.
[8] S. Adya and I. Markov. Fixed-outline floorplanning : Enabling hierarchical design. *IEEE Trans. on VLSI Systems*, 11(6):1120–1135, December 2003.
[9] S. N. Adya, S. Chaturvedi, J. A. Roy, D. A. Papa, and I. L. Markov. Unification of partitioning, placement and floorplanning. In *Proc. of ICCAD*, pages 550–557, 2004.
[10] C. J. Alpert, J.-H. Huang, and A. B. Kahng. Multilevel circuit partitioning. *IEEE Trans. on CAD*, 17(8):655–667, August 1998.

TABLE III

COMPARISONS FOR HPWL, DEAD SPACE (DS), AND CPU TIME AMONG B*-TREE, MB*-TREE, PARQUET, IMF, IMF+AFF, AND CAPO. THE AFF STANDS FOR ACCELERATIVE FIXED-OUTLINE FLOORPLANNING. *THE HPWL VALUES WITH PARENTHESES DENOTE THAT THE RESULTS CANNOT FIT INTO THE BOUNDING BOXES (B*-TREE/MB*-TREE), OR THERE ARE OVERLAPS IN THE RESULT (CAPO 9). THE AVERAGE HPWL RATIO CONSIDERS ONLY THE RESULTS THAT CAN FIT INTO THE BOUNDING BOXES.

Circuit	#Module /#Net	Variable-die floorplanner				Fixed-die floorplanner						Fixed-die floorplacer	
		B*-tree WireOpt		MB*-tree WireOpt		Best of Parquet 3.1/4.0		IMF (Ours)		IMF+AFF (Ours)		Capo 9	
		HPWL /DS(%)	Time (sec)	HPWL /DS(%)	Time (sec)	HPWL	Time (sec)	HPWL	Time (sec)	HPWL	Time (sec)	HPWL	Time (sec)
apte	9/97	*(409678)/30.4	0.1	*(409678)/15.2	0.2	447958	0.2	425322	0.7	500058	0.1	548972	0.2
xerox	10/203	*(486729)/9.9	0.3	*(491502)/26.1	0.2	524386	0.6	505394	0.9	555752	0.2	530526	1.2
hp	11/83	*(117212)/24.8	0.2	*(118364)/44.5	0.2	130120	0.4	124085	0.7	174559	0.1	*(130177)	1.2
ami33	33/123	*(53282)/32.1	1.3	*(53219)/24.8	2.2	68023	1.7	62033	2.0	65231	0.8	73164	3.4
ami49	49/408	*(763734)/31.7	5.7	*(722894)/30.1	3.9	882537	5.5	867663	5.4	947897	1.2	1001170	3.9
n10	10/118	*(34750)/24.9	0.1	*(34977)/18.7	0.2	38034	0.3	36068	0.5	39167	0.2	41933	0.4
n30	30/349	*(104523)/15.6	1.2	*(105930)/25.8	2.5	114462	2.3	106910	1.7	108435	0.6	109864	0.8
n50	50/485	*(132482)/20.9	3.9	*(136316)/20.0	9.3	146808	4.6	134368	7.0	135828	1.5	141116	2.3
n100	100/885	*(204193)/21.1	21.0	*(214036)/28.0	37.5	242050	17.5	207852	11.5	208772	2.3	224390	4.6
n200	200/1585	*(375912)/25.7	94.9	*(404963)/28.0	66.3	432882	77.2	369888	64.6	372845	4.2	385594	15.0
n300	300/1893	*(519250)/27.9	194.9	*(613444)/26.5	85.5	647452	166.2	489868	91.7	494480	5.5	522968	13.5
Avg. HPWL Ratio		*(0.96)/24.1		*(0.99)/26.2		1.10		1.00		1.09		1.11	
Total Time			323.7		207.9		276.5		186.7		16.7		45.3

TABLE IV

COMPARISONS FOR HPWL, DEAD SPACE (DS), AND CPU TIME AMONG B*-TREE, MB*-TREE, PARQUET, IMF+AFF, AND CAPO FOR `ami49_x` CIRCUITS. *THE HPWL VALUES WITH PARENTHESES DENOTE THAT THE RESULTS CANNOT FIT INTO THE BOUNDING BOXES. NR: NO RESULT OBTAINED WITHIN 12-HR CPU TIME.

Circuit	#Module /#Net	Variable-die floorplanner						Fixed-die floorplanner				Fixed-die floorplacer	
		B*-tree WireOpt			MB*-tree WireOpt			Best of Parquet 3.1/4.0		IMF+AFF (Ours)		Capo 9	
		HPWL (× e6)	DS (%)	Time (min)	HPWL (× e6)	DS (%)	Time (min)	HPWL (× e6)	Time (min)	HPWL (× e6)	Time (min)	HPWL (× e6)	Time (min)
ami49_1	49/408	*(0.16)	32.43	0.1	*(0.15)	30.85	0.1	0.19	0.1	0.18	0.0	0.20	0.0
ami49_2	98/865	*(0.46)	37.44	0.4	*(0.39)	28.75	0.4	0.54	0.3	0.47	0.0	0.54	0.1
ami49_4	196/1779	*(1.16)	43.65	2.0	*(1.06)	31.20	0.8	1.56	1.3	1.07	0.1	1.14	0.3
ami49_10	490/4521	*(4.28)	36.66	16.9	*(3.79)	39.22	2.1	6.03	9.6	3.31	0.2	3.48	0.7
ami49_20	980/9091	*(1.18)	39.88	94.1	*(1.02)	44.73	4.8	16.12	44.5	8.13	0.5	8.48	1.5
ami49_40	1960/18231	*(3.08)	37.05	609.1	*(2.67)	50.24	12.7	40.39	272.9	2.02	1.3	2.07	3.3
ami49_60	2940/27371	NR	NR	NR	*(4.99)	48.64	25.7	74.14	734.4	3.42	2.2	3.54	5.5
ami49_80	3920/36511	NR	NR	NR	*(7.48)	55.28	36.7	NR	NR	5.05	3.5	5.22	8.8
ami49_100	4900/45651	NR	NR	NR	*(10.29)	54.18	66.2	NR	NR	6.86	5.1	7.13	10.6
ami49_150	7350/68501	NR	NR	NR	*(18.44)	56.08	107.5	NR	NR	12.08	11.0	12.40	13.7
ami49_200	9800/91351	NR	NR	NR	*(33.39)	67.80	224.8	NR	NR	18.00	19.1	18.33	19.8
Avg. HPWL Ratio		*(1.20)			*(1.29)			1.56		1.00		1.05	

[11] H. H. Chan, S. N. Adya, and I. L. Markov. Are floorplan representations important in digital design? In *Proc. of ISPD*, pages 129–136, 2005.

[12] T. Chan, J. Cong, T. Kong, and J. R. Shinnerl. Multilevel optimization for large-scale circuit placement. In *Proc. of ICCAD*, pages 171–176, 2000.

[13] C.-C. Chang, J. Cong, and X. Yuan. Multi-level placement for large-scale mixed-size ic designs. In *Proc. of ASPDAC*, pages 325–330, 2003.

[14] 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 DAC*, pages 458–463, 2000.

[15] T.-C. Chen and Y.-W. Chang. Modern floorplanning based on fast simulated annealing. In *Proc. of ISPD*, pages 104–112, 2005.

[16] 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 ISPD*, pages 236–238, 2005.

[17] J. Cong, J. Fang, and Y. Zhang. Multilevel approach to full-chip gridless routing. In *Proc. of ICCAD*, pages 396–403, 2001.

[18] J. Cong, M. Romesis, and J. R. Shinnerl. Fast floorplanning by look-ahead enabled recursive bipartitioning. In *Proc. of ASPDAC*, 2005.

[19] J. Cong, M. Xie, and Y. Zhang. An enhanced multilevel routing system. In *Proc. of ICCAD*, pages 51–58, 2002.

[20] P.-N. Guo, C.-K. Cheng, and T. Yoshimura. An O-tree representation of non-slicing floorplan and its applications. In *Proc. of DAC*, pages 268–273, 1999.

[21] B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graph. In *Proc. of Supercomputing*, 1995.

[22] T.-Y. Ho, Y.-W. Chang, S.-J. Chen, and D. T. Lee. A fast crosstalk- and performance-driven multilevel routing system. In *Proc. of ICCAD*, pages 382–387, 2003.

[23] X. Hong, G. Huang, T. Cai, J. Gu, S. Dong, C.-K. Cheng, and J. Gu. Corner block list: An effective and efficient topological representation of non-slicing floorplan. In *Proc. of ICCAD*, pages 8–12, 2000.

[24] C.-C. Hu, D.-S. Chen, and Y.-W. Wang. Fast multilevel floorplanning for large scale modules. In *International Symposium on Circuits and Systems*, pages 205–208, 2004.

[25] A. B. Kahng. Classical floorplaning harmful? In *Proc. of ISPD*, pages 207–213, 2000.

[26] A. B. Kahng and S. Reda. Placement feedback: a concept and method for better min-cut placements. In *Proc. of DAC*, pages 357–362, 2004.

[27] G. Karypis and V. Kumar. Multilevel k-way hypergraph partitioning. In *Proc. of DAC*, pages 343–348, 1999.

[28] H.-C. Lee, Y.-W. Chang, J.-M. Hsu, and H. H. Yang. Multilevel floorplanning/placement for large-scale modules using B*-trees. In *Proc. of DAC*, pages 812–817, 2003.

[29] J.-M. Lin and Y.-W. Chang. TCG: A transitive closure graph-based representation for non-slicing floorplans. In *Proc. of DAC*, pages 764–769, 2001.

[30] J.-M. Lin and Y.-W. Chang. TCG-S: Orthogonal coupling of P*-admissible representations for general floorplans. In *Proc. of DAC*, pages 842–847, 2002.

[31] J.-M. Lin, Y.-W. Chang, and S.-P. Lin. Corner sequence - A P-admissible floorplan representation with a worst case linear-time packing scheme. *IEEE Trans. on VLSI Systems*, 11(4):679–686, August 2003.

[32] S.-P. Lin and Y.-W. Chang. A novel framework for multilevel routing considering routability and performance. In *Proc. of ICCAD*, pages 44–50, 2002.

[33] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajatani. Rectangle-packing based module placement. In *Proc. of ICCAD*, pages 472–479, 1995.

[34] S. Nakatake, K. Fujiyoshi, H. Murata, and Y. Kajatani. Module placement on bsg-structure and ic layout applications. In *Proc. of ICCAD*, pages 484–491, 1996.

[35] N. Selvakkumaran and G. Karypis. Theto - a fast and high-quality paritioning driven global placer. Technical Report 03-46, Dept of Computer Science and Engineering, Univeristy of Minnesota, Novemver 2003.

[36] N. Selvakkumaran and G. Karypis. Theto - a fast and high-quality paritioning driven placement tool. Technical Report 04-40, Dept of Computer Science and Engineering, Univeristy of Minnesota, October 2004.

[37] E. F. Y. Young, C. C. N. Chu, and Z. C. Shen. Twin binary sequences: A nonredundant representation for general nonslicing floorplan. *IEEE Trans. on VLSI Systems*, 22(4):457–469, April 2003.