DeFer: Deferred Decision Making Enabled Fixed-Outline Floorplanning Algorithm

Jackey Z. Yan and Chris Chu

***Abstract*—In this paper, we present *DeFer*—a fast, high-quality, scalable, and nonstochastic fixed-outline floorplanning algorithm. *DeFer* generates a nonslicing floorplan by compacting a slicing floorplan. To find a good slicing floorplan, instead of searching through numerous slicing trees by simulated annealing as in traditional approaches, *DeFer* considers only one *single* slicing tree. However, we generalize the notion of slicing tree based on the principle of deferred decision making (DDM). When two subfloorplans are combined at each node of the generalized slicing tree, *DeFer* does not specify their orientations, the left–right/top–bottom order between them, and the slice line direction. *DeFer* even does not specify the slicing tree structure for small subfloorplan. In other words, we are deferring the decisions on these factors, which are specified arbitrarily at an early step in traditional approaches. Because of DDM, one slicing tree actually corresponds to a large number of slicing floorplan solutions, all of which are efficiently maintained in one *single* shape curve. With the final shape curve, it is straightforward to choose a good floorplan fitting into the fixed outline. Several techniques are also proposed to further optimize the wirelength. For both fixed-outline and classical floorplanning problems, experimental results show that *DeFer* achieves the *best* success rate, the *best* wirelength, the *best* runtime, and the *best* area on average compared with all other state-of-the-art floorplanners.**

***Index Terms*—Deferred decision making, fixed outline, floorplanning, layout optimization.**

I. Introduction

FLOORPLANNING has become a very crucial step in modern very large scale integration (VLSI) designs. As the start of physical design flow, floorplanning not only determines the top-level spatial structure of a chip, but also initially optimizes the interconnections. Thus, a good floorplan solution among circuit modules definitely has a positive impact on the placement, routing, and even manufacturing. In the nanometer scale era, the ever-increasing complexity of integrated circuits (ICs) promotes the prevalence of hierarchical design. However, as pointed out by Kahng [1], classical outline-free floorplanning [2] cannot satisfy such requirements of modern designs. In contrast with this, fixed-outline floorplanning enabling the hierarchical framework is preferred by modern application-specific integrated circuit designs. Nevertheless, fixed-outline floorplanning has been shown to be much more difficult, compared with classical outline-free floorplanning, even without considering wirelength optimization [3].

Manuscript received January 31, 2009; revised May 31, 2009 and October 7, 2009. Current version published February 24, 2010. This work was partially supported by International Business Machines, Faculty Award, and National Science Foundation under Grant CCF-0540998. This paper was recommended by Associate Editor L. Scheffer.

The authors are with the Department of Electrical and Computer Engineering, Iowa State University, Ames, IA 50010 USA (e-mail: zijunyan@iastate.edu; cnchu@iastate.edu).

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.2010.2041850

A. Previous Work

Simulated annealing has been the most popular method of exploring good solutions on the fixed-outline floorplanning problem. Using sequence pair representation, Adya *et al.* [4] modified the objective function, and proposed a few new moves based on slack computation to guide a better local search. To improve the floorplanning scalability and initially optimize the interconnections, in [5] the original circuit is first cut into multiple partitions by a min-cut partitioner. Simultaneously, the chip region is split into small bins. After that, the annealing-based floorplanner [4] performs fixed-outline floorplanning on each partition within its associated bin. In [6], Chen *et al.* adopted the B*-tree [7] representation to describe the geometric relationships among modules, and performed a novel three-stage cooling schedule to speed up the annealing process. In [8] a multilevel partitioning step is performed beforehand on the original circuit. Different from [5], the annealing-based fixed-outline floorplanner is performed iteratively at each level of the multilevel framework. By enumerating the positions in sequence pairs, Chen *et al.* [9] applied insertion after remove (IAR) to accelerate the simulated annealing. As a result, both the runtime and success rate[1] are enhanced dramatically. Recently, using Ordered Quadtree representation, He *et al.* [10] adopted quadratic equations to solve the fixed-outline floorplanning problem.

All of the above techniques are based on simulated annealing. Generally, the authors tried various approaches to improve the algorithm efficiency. But one common drawback is that these techniques do not have a good scalability. They become quite slow when the size of circuits grows large, e.g., 100 modules. Additionally, the annealing-based techniques always have a hard time handling circuits with soft modules, because they need to search a large solution space, which can be time-consuming.

Some researchers have adopted nonstochastic methods. In [11], a slicing tree is first built up by recursively partitioning the original circuit until each leaf node contains at most

[1]*Success rate* is defined as the ratio of the number of runs resulting a layout within fixed die, to the total number of runs.

two modules. Then the authors rely on various heuristics to determine the geometry relationships among the modules and output a final floorplan solution. Sassone *et al.* [12] proposed an algorithm containing two phases. First the modules are grouped together only based on connectivity. Second the modules are packed physically by a row-oriented block packing (ROB) technique which organizes the modules by rows based on their dimensions. But this technique cannot handle soft modules. In [13], Zhan *et al.* applied a quadratic analytical approach similar to those used for placement problems. To generate a nonoverlapping floorplan, the quadratic approach relies on a legalization process. However, this legalization is very difficult for circuits with big hard macros. Cong *et al.* [14] presented an area-driven look-ahead floorplanner in a hierarchical framework. Two main techniques are used in their algorithm: the ROB and zero-dead space (ZDS). To handle both hard and soft modules, ROB is extended from [12]. ZDS is used to pack soft modules. But, ROB may generate a layout with large whitespace when the module sizes in a subfloorplan are quite different from each other, e.g., a design with big hard macros.

B. Our Contributions

This paper presents a fast, high-quality, scalable, and nonstochastic fixed-outline floorplanner called *DeFer*.[2] It can efficiently handle both hard and soft modules.

DeFer generates a final nonslicing floorplan by compacting a slicing floorplan. It has been proved in [16] that any nonslicing floorplan can be generated by compacting a slicing floorplan. In traditional annealing-based approaches, obtaining a good slicing floorplan usually takes a long time, because the algorithms have to search many slicing trees. By comparison, *DeFer* considers only one *single* slicing tree generated by recursive partitioning. However, to guarantee that a large solution space is explored, we generalize the notion of slicing tree [2] based on the principle of deferred decision making (DDM). When two subfloorplans are combined at each node of the generalized slicing tree, *DeFer* does not specify their orientations, the left–right/top–bottom order between them, and the slice line direction. For small subfloorplan, *DeFer* even does not specify its slicing tree structure, i.e., the skeletal structure (not including tree nodes) in the slicing tree. In other words, we are deferring the decisions on these four factors correspondingly: 1) subfloorplan orientation; 2) subfloorplan order; 3) slice line direction; and 4) slicing tree structure. Because of DDM, one slicing tree actually represents a large number of slicing floorplan solutions. In *DeFer*, all of these solutions are efficiently maintained in a *single* shape curve [17]. With the final shape curve, it is straightforward to choose a good slicing floorplan fitting into the fixed outline. To realize the DDM idea, we propose the following techniques.

- *Generalized Slicing Tree:* To defer the decisions on these three factors: 1) subfloorplan orientation; 2) subfloorplan order; and 3) slice line direction, we generalize the original slicing tree. In the generalized slicing tree, one tree node can represent both orientations of its two child nodes, both orders between them, and both horizontal and vertical slice lines. Note that the work in [17] and [18] only generalized the orientation for *individual module* and the slice line direction, respectively. In order to carry out the combination of generalized slicing trees, we also extend original shape curve operation to curve *Flipping* and curve *Merging*.[3]
- *Enumerative Packing:* To defer the decision on the slicing tree structure within small subfloorplan, we develop the enumerative packing (EP) technique. It enumerates all possible slicing structures, and builds up one shape curve capturing all slicing layouts among the modules of small subfloorplan. The naive enumeration is very expensive in terms of CPU time and memory usage. But using the technique of dynamic programming, EP can be efficiently applied to up to 10 modules.
- *Block Swapping and Mirroring:* To make the decision on the subfloorplan order (left–right/top–bottom), we adopt three techniques: *Rough Swapping*, *Detailed Swapping* [11], and *Mirroring*. The motivation is to greedily optimize the wirelength. As far as we know, we are the first proposing the *Rough Swapping* technique and showing that without *Rough Swapping Detailed Swapping* may degrade the wirelength.

Additionally, we adopt the following three methods to enhance the robustness and quality of *DeFer*.

- *Terminal Propagation (TP): DeFer* accounts for fixed pins by using TP [19] during partitioning process.
- *Whitespace-Aware Pruning (WAP):* A pruning method is proposed to systematically control the number of points on each shape curve.
- *High-Level EP:* Based on EP, we propose the high-level EP technique to further improve the packing quality.

By switching the strategy of selecting the points on the final shape curve, we extend *DeFer* to handle other floorplanning problems, e.g., classical outline-free floorplanning,

For fixed-outline floorplanning, experimental results on *GigaScale Systems Research Center (GSRC) Hard-Block*, *GSRC Soft-Block*, *hybrid blocks (HB)* (containing both hard and soft modules), and *HB+* (a hard version of *HB*) benchmarks show that *DeFer* achieves the *best* success rate, the *best* wirelength, and the *best* runtime on average, compared with all other state-of-the-art floorplanners. The runtime difference between small and large circuits shows *DeFer's* good scalability. For classical outline-free floorplanning, using a linear combination of area and wirelength as the objective, *DeFer* achieves 12% better cost value than *Parquet 4.5* with 76× faster runtime.

The rest of this paper is organized as follows. Section II describes the algorithm flow. Section III introduces the *Generalized Slicing Tree*. Section IV describes the *Whitespace-Aware Pruning*. Section V describes the *Enumerative Packing* technique. Section VI illustrates the *Block Swapping and Mirroring*. Section VII introduces the extension of *DeFer* on other floorplanning problems. Section VIII addresses the implementation details. Experimental results are

[2]A preliminary version of *DeFer* was presented in [15].

[3]In this paper, all *slicing trees* and *shape curve operation* stand for the generalized version by default.

Algorithm Flow of *DeFer*
Begin
Step 1): Top-down recursive min-cut bisectioning
Step 2): Bottom-up recursive shape curve combination
Step 3): Top-down tracing selected points
Step 4): Top-down wirelength refinement by swapping
Step 5): Slicing floorplan compaction
Step 6): Greedy wirelength-driven shifting
End

Fig. 1. Pseudocode on algorithm flow of *DeFer*.

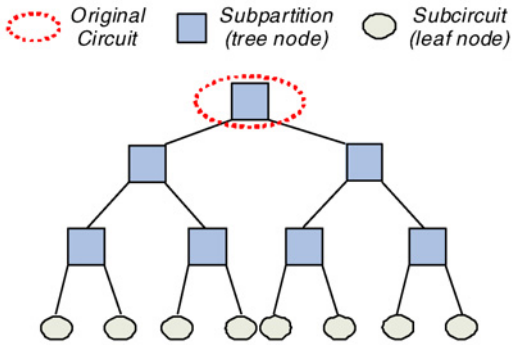

Fig. 2. High-level slicing tree.

presented in Section IX. Finally, this paper ends with a conclusion.

II. Algorithm Flow of DeFer

Essentially, *DeFer* has six steps as shown in Fig. 1. The details of each step are as follows.

1) *Partitioning Step:* As the number of modules in one design becomes large, exploring all slicing layout solutions among them is very expensive. Thus, the purpose of this step is to divide the original circuit into several small subcircuits, and initially minimize the interconnections among them. *hMetis* [20], the state-of-the-art hypergraph partitioner, is called to perform a recursive bisectioning on the circuit, until every partition contains less than or equal to *maxN* modules (*maxN* = 10 by default). TP is used in this step. Theoretically TP can be applied at any cut. But as using TP degrades the packing quality (see Section III-C), we apply it only at the first cut on the original circuit. During partitioning, a high-level slicing tree structure is built up where each leaf node is a *subcircuit*, and each tree node is a *subpartition* (see Fig. 2). Due to the generalized notion of slicing tree, the whole high-level slicing tree not only sets up a hierarchical framework, but also represents many possible packing solutions among the subcircuits.
2) *Combining Step:* In this step, we first defer the decision on the slicing tree structure of each subcircuit, by applying the *Enumerative Packing* technique to explore all slicing packing layouts within the subcircuit. After that, an associated shape curve representing these possible layouts for each subcircuit is produced. Then, based on the hierarchical framework in Step 1, *DeFer* traverses from bottom-up constructing a shape curve for every tree node. The final shape curve at the root will maintain all explored slicing floorplan layouts of the whole circuit.
3) *Back-Tracing Step:* Once the final shape curve is available, it is fairly straightforward to select the points fitting into the fixed outline (see Fig. 3). For each of the points we select, a back-tracing[4] process is applied. As every point in the parent curve is generated by adding two points from two child curves, basically the back-tracing is to trace the selected point on each shape curve from top-down. During this process, *DeFer* makes the decisions on every subfloorplan orientation, slice line direction, and slicing tree structure of each subcircuit.
4) *Swapping Step:* The fourth step is to make decisions on the subfloorplan order (left–right/top–bottom), by greedily swapping every two child subfloorplans. Basically we perform three wirelength refinement processes through the hierarchical framework. First, *Rough Swapping* is applied from top-down, followed by *Detailed Swapping*. Finally, we apply *Mirroring*.
5) *Compacting Step:* After fixing the slicing floorplan, this step is to compact all modules to the center of the fixed outline. The compaction puts modules closer to each other, such that the wirelength is further reduced. If the slicing floorplan is outside of the fixed outline, *DeFer* compacts them to the lower-left corner rather than the center, so that potentially there is a higher chance to find a valid layout within the fixed outline.
6) *Shifting Step:* In Step 5, some modules may be over-compacted. So we greedily shift such modules toward the optimal positions [21] regarding wirelength minimization. At the end, *DeFer* outputs the final floorplan.

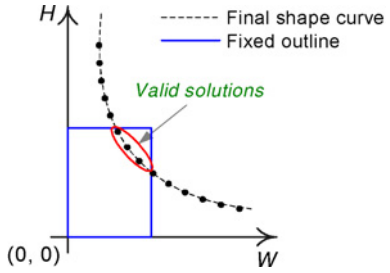

Fig. 3. Final shape curve with fixed outline and candidate points.

From the algorithm flow, we can see that by initially *deferring* the decisions in Steps 1 and 2, *DeFer* explores a large collection of slicing layouts, all of which are efficiently maintained in one final shape curve at the top; by finally *making* the decisions in Steps 3 and 4, *DeFer* chooses good slicing layouts fitting into the fixed outline. The main techniques are discussed in detail in Sections III–VII.

[4]*Back-tracing* is different from *back-tracking* [5] which traverses from bottom-up to determine legal solutions.

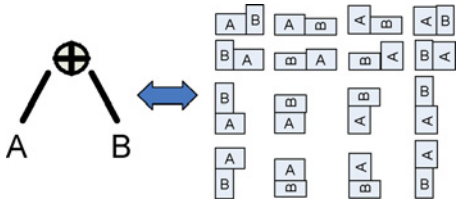

Fig. 4. Generalized slicing tree and sixteen different layouts.

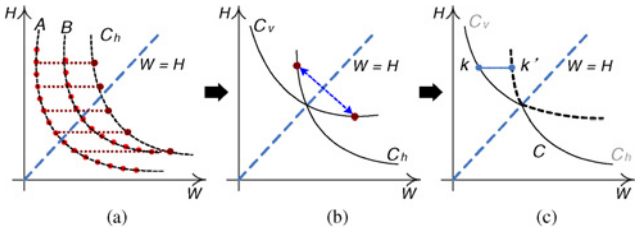

Fig. 5. Extended shape curve operation. (a) Addition. (b) Flipping. (c) Merging.

III. Generalized Slicing Tree

In this section, we introduce the generalized slicing tree, which enables the deferred decisions on these three factors: 1) subfloorplan orientation; 2) subfloorplan order; and 3) slice line direction.

A. Notion of Generalized Slicing Tree

In an ordinary slicing tree, the parent tree node of two child subfloorplans A and B is labeled "H"/"V" to specify that A and B are separated by a horizontal/vertical slice line, and the order between the two child nodes in the slicing tree specifies the top–bottom/left–right order of A and B in the layout. For example, if in the ordinary slicing tree the left child is A, the right child is B, and the parent node is labeled "V," then in the corresponding layout A is on the left of B. If we want to switch to other layouts between A and B, then the slicing tree has to be changed as well.

Now we generalize the ordinary slicing tree, such that one generalized slicing tree represents multiple slicing layouts. Here, we introduce a new operator—"$\oplus$" to incorporate both "H" and "V" slice line directions. Moreover, we do not differentiate the "top–bottom" or "left–right" order between the two child subfloorplans any more, which means even though we put A at the left child, it can be switched to the right later on. We even do not specify the orientation for each subfloorplan. As a result, the decisions on slice line direction, subfloorplan order, and subfloorplan orientation are deferred. Now each parent node in the slicing tree represents *all sixteen* slicing layouts between two child subfloorplans (see Fig. 4).

B. Extended Shape Curve Operation

To actualize the slicing tree combination we use the shape curve operation. The shape of each subfloorplan is captured by its associated shape curve. In order to derive a compatible operation for the new operator "$\oplus$," we develop three steps to combine two child curves A and B into one parent curve C.

1) *Addition:* Firstly, we add two curves A and B horizontally to get curve C_h, on which each point corresponds to a horizontal combination of two subfloorplan layouts from A and B, respectively [see Fig. 5(a)].
2) *Flipping:* Next, we flip curve C_h symmetrically based on the $W = H$ line to derive curve C_v. The purpose of doing this is to generate the curve that contains the corresponding vertical combination cases from the two subfloorplan layouts [see Fig. 5(b)].
3) *Merging:* The final step is to merge C_h and C_v into the parent curve C. Since the curve function is a *bijection* from set W to set H, for a given height only one point can be kept. We choose the point with a smaller width out of C_h and C_v, e.g., point k in Fig. 5(c), because such point corresponds to smaller floorplan area.

As a result, we have derived three steps to actualize the operator "$\oplus$" in the slicing tree combination. Now given two child curves corresponding to two child subfloorplans in the slicing tree, these three steps are applied to combine the two curves into one parent curve, in which the entire slicing layouts between the two child subfloorplans are captured.

C. Decision of Slice Line Direction for Terminal Propagation

Because all cut line directions in the high-level slicing tree are undetermined, we cannot apply TP during partitioning. In order to enable TP, we *pre-decide* the cut line direction based on the aspect ratio[5] τ_p of the subpartition region. That is, if $\tau_p > 1$, the subpartition will be cut "horizontally;" otherwise, it will be cut "vertically." In principle, we can use such strategy on all cut lines in the high-level slicing tree. However, by doing this we restrict the combine direction in the generalized slicing tree, which degrades the packing quality. To make a trade-off, we only apply TP at the root, i.e., the first cut on the original circuit.

IV. Whitespace-Aware Pruning

In this section, we present the WAP technique, which systematically prunes the points on the shape curve with whitespace awareness.

A. Motivation on WAP

In Fig. 6, two subfloorplans A and B are combined into subfloorplan C. Shape curves C_a, C_b, and C_c contain various floorplan solutions of A, B, and C, respectively. Because C_b has a gap between points P_2 and P_3, during the combining process point P_1 cannot find any point from C_b with the matched height, and is forced to combined with P_2. Due to the height difference between P_1 and P_2, the resulted point P_4 on curve C_c represents a layout with extra whitesapce. The bigger the gap is, the more the whitespace is generated.

It is only an ideal situation that each point always had a matched point on another curve. Therefore, in the hierarchical framework during the curve combining process, the whitespace will be generated and accumulated to the top level. For a fixed-outline floorplanning problem, we have a budget/maximum

[5]In this paper, *aspect ratio* is defined as the ratio of height to width.

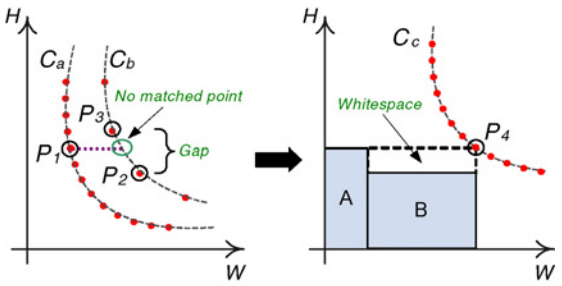

Fig. 6. Generation of whitespace during curve combination.

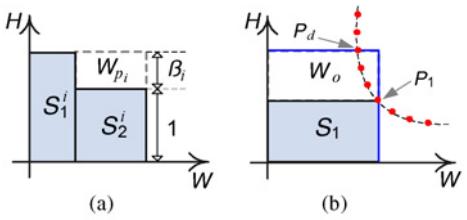

Fig. 7. Calculation of W_{p_i} and W_o.

whitespace amount W_b. In order to avoid exceeding W_b, the whitespace generated in the curve combination needs to be minimized. One direct way to achieve this is to increase the number of points, such that the sizes of gaps among the points are minimized. However, the more points we keep, the slower the algorithm runs. This rises the question WAP is trying to solve: *How can we minimize the number of points on the shape curve, while guaranteeing that the total whitespace would not exceed W_b?*

B. Problem Formulation of WAP

WAP is to prune the points on the shape curve, while making sure that the gaps among the points are small enough, such that we can guarantee the total whitespace would not exceed the budget W_b. WAP is formulated as follows:

$$\begin{aligned} \text{Minimize} \quad & \sum_{i=1}^{M} k_i \\ \text{subject to} \quad & \sum_{i=1}^{M} W_{p_i} + \sum_{j=1}^{N} W_{c_j} + W_o \le W_b. \end{aligned} \tag{1}$$

In (1), suppose there are M subpartitions and N subcircuits in the high-level slicing tree (see Fig. 2). Before pruning, there are k_i points on shape curve i of subpartition i. During the combine process of generating shape curve i, the introduced whitespace in subpartition i is W_{p_i}. The whitespace inside subcircuit j is W_{c_j}. At the root, the whitespace between the floorplan outline and the fixed outline is W_o.

To do pruning, we calculate a pruning parameter β_i for shape curve i. In subpartition i, let the corresponding width and height of point p ($1 \le p \le k_i$) be w_p^i and h_p^i. On each shape curve, the points are sorted based on the ascending order of the height. ΔH_p is defined for point p as follows:

$$\Delta H_p = \beta_i \cdot h_p^i. \tag{2}$$

Within the distance of ΔH_p above point p, only the point that is the closest to $h_p^i + \Delta H_p$ is kept, and other points are pruned away. The intuition is that the gap within ΔH_p is small enough to guarantee that no large whitespace will be generated. Such pruning method is applied only on every pair of child curves of *subpartitions* in the high-level slicing tree, before they are combined into a parent curve. We do not prune any point on the shape curves of *subcircuits*.

Now we rewrite (1) into a form related with β_i, such that by solving WAP we can get the value of β_i. Based on the above pruning, we have $h_{p+1}^i \le (1+\beta_i) \cdot h_p^i$. So approximately $h_{p+2}^i \ge (1+\beta_i) h_p^i$. Thus, the relationship between the first point and point k_i is

$$h_{k_i}^i \ge (1+\beta_i)^{\frac{k_i-1}{2}} h_1^i \Rightarrow k_i \le 2 \cdot \left(\frac{\ln(h_{k_i}^i / h_1^i)}{\ln(1+\beta_i)} \right) + 1. \tag{3}$$

Because of the Flipping [see Fig. 5(b)], each shape curve is symmetrical based on $W = H$ line. So in the implementation we only keep the lower half curve. In this case, the last point k_i is actually very close[6] to $W = H$ line, so we have

$$w_{k_i}^i \approx h_{k_i}^i \Rightarrow h_{k_i}^i \approx \sqrt{A_i} \tag{4}$$

where A_i is the area of subpartition i. It equals to the sum of total module area in subpartition i and the accumulated whitesapce from the subcircuits at lower level. In (3), h_1^i is actually the minimum height of the outlines on shape curve i. Suppose subpartition i contains V_i modules. The width and height of module m are x_m^i and y_m^i

$$h_1^i = \max(\min(x_1^i, y_1^i), \ldots, \min(x_{V_i}^i, y_{V_i}^i)). \tag{5}$$

In the following part, we explain the calculation of other terms in (1).

- *Calculation of W_{p_i}:* Suppose two child subpartitions S_1^i and S_2^i are combined into parent subpartition S_i, where the area of S_1^i, S_2^i and S_i are A_1^i, A_2^i and A_i. The pruning parameter of S_i is β_i. As shown in Fig. 7(a), the whitespace produced in the combining process is

$$W_{p_i} = A_i \cdot \frac{A_2^i \cdot \beta_i}{A_1^i + A_2^i + A_2^i \cdot \beta_i}. \tag{6}$$

  Since the partitioner tries to balance the area of S_1^i and S_2^i, we can assume $A_1^i \approx A_2^i$. Typically $\beta_i \ll 2$, so $A_1^i + A_2^i + A_2^i \cdot \beta_i \approx A_i$. Thus

$$W_{p_i} = A_1^i \cdot \beta_i = A_2^i \cdot \beta_i = A_i \cdot \frac{\beta_i}{2}. \tag{7}$$

- *Calculation of W_{c_j}:* Before pruning, the shape curves of subcircuits have already been generated by EP. We choose the minimum whitespace among all layouts of subcircuit j as the value of W_{c_j}, so that $\sum_{j=1}^{N} W_{c_j} \ge W_b$ can be prevented.
- *Calculation of W_o:* At the root, there is extra whitespace W_o between the floorplan outline and the fixed outline. *DeFer* picks at most δ points ($\delta = 21$ by default) for back-tracing step. So we assume there are δ points enclosed

[6]If k_i represents a outline of a square, it is on $W = H$ line.

into the fixed outline, and the first and last points P_1, P_d out of δ are on the right and top boundary of the fixed outline [see Fig. 7(b)]. For various points/layouts, W_o is different. We use the one of P_1 to approximate W_o. As in pruning we always keep the point that is the closest to $(1+\beta_i)h_p^i$, here we can assume $h_{p+1}^1 = (1+\beta_1)h_p^1$. So we have

$$W_o = A_1 \cdot ((1+\beta_1)^{\delta-1} - 1). \tag{8}$$

From (3), (4), (7), and (8), (1) can be rewritten as

$$\begin{aligned} \text{Minimize} \quad & \sum_{i=1}^{M} \frac{\ln(\sqrt{A_i}/h_1^i)}{\ln(1+\beta_i)} \\ \text{subject to} \quad & \sum_{i=1}^{M} A_i \cdot \frac{\beta_i}{2} + \sum_{j=1}^{N} W_{c_j} + W_o \le W_b \\ & W_o = A_1 \cdot ((1+\beta_1)^{\delta-1} - 1) \\ & \beta_i \ge 0 \quad i = 1, \ldots, M. \end{aligned} \tag{9}$$

C. Solving WAP

To solve WAP (9), we relax the constraint related with W_b by Lagrangian relaxation. Let λ be the nonnegative Lagrange multiplier, and $W' = W_b - \sum_{j=1}^{N} W_{c_j} - W_o$

$$L_\lambda(\beta_i) = \sum_{i=1}^{M} \frac{\ln(\sqrt{A_i}/h_1^i)}{\ln(1+\beta_i)} + \lambda \cdot (\sum_{i=1}^{M} A_i \cdot \frac{\beta_i}{2} - W')$$

$$\begin{aligned} \textit{LRS}: \text{Minimize} \quad & L_\lambda(\beta_i) \\ \text{subject to} \quad & \beta_i \ge 0 \quad i = 1, \ldots, M. \end{aligned}$$

LRS is the Lagrangian relaxation subproblem associated with λ. Let the function $Q(\lambda)$ be the optimal value of *LRS*. The Lagrangian dual problem (*LDP*) is defined as

$$\begin{aligned} \textit{LDP}: \text{Maximize} \quad & Q(\lambda) \\ \text{subject to} \quad & \lambda \ge 0. \end{aligned}$$

As WAP is a convex problem, if λ is the optimal solution of *LDP*, then the optimal solution of *LRS* also optimizes WAP. We differentiate $L_\lambda(\beta_i)$ based on β_i and λ, respectively

$$\frac{\partial L}{\partial \beta_1} = \lambda A_1 \left(\frac{1}{2} + (\delta - 1) \cdot ((1+\beta_1)^{\delta-2}) \right) - \frac{\ln(\sqrt{A_1}/h_1^1)}{(1+\beta_1) \cdot \ln^2(1+\beta_1)}.$$

$$\frac{\partial L}{\partial \beta_i} = \frac{\lambda A_i}{2} - \frac{\ln(\sqrt{A_i}/h_1^i)}{(1+\beta_i) \cdot \ln^2(1+\beta_i)}, \quad i = 2, \ldots, M.$$

$$\frac{\partial L}{\partial \lambda} = \sum_{i=1}^{M} A_i \cdot \frac{\beta_i}{2} - W'.$$

To find the "saddle point" between *LRS* and *LDP*, we first set an arbitrary λ. Once λ is fixed, $\frac{\partial L}{\partial \beta_i}$ $(1 \le i \le M)$ is a *univariate* function that can be solved by *Bisection Method* to get β_i. Then β_i is used to get the value of function $\frac{\partial L}{\partial \lambda}$. If $\frac{\partial L}{\partial \lambda} \neq 0$, we adjust λ accordingly based on *Bisection Method* and do another iteration of the above calculation, until $\frac{\partial L}{\partial \lambda} = 0$.

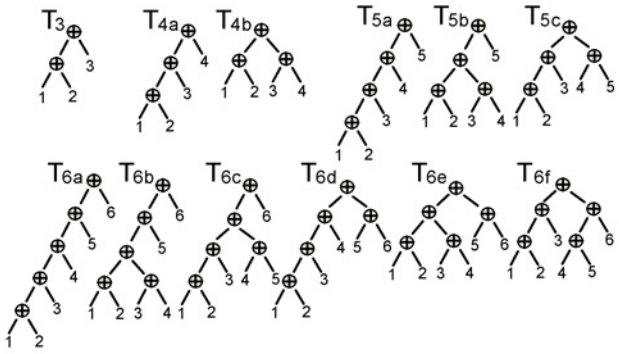

Fig. 8. List of different slicing tree structures.

Eventually, the pruning parameters β_i returned by WAP are used to systematically prune the points on the shape curve of each subpartition i. Best of all, we do not need to worry about the over-pruning and degradation of the packing quality.

V. Enumerative Packing

In order to defer the decision on the slicing tree structure, we propose the EP technique that can efficiently enumerate all possible slicing layouts among a set of modules, and finally keep all of them into one shape curve.

A. Naive Approach of Enumeration

In this section, we plot out a naive way to enumerate all slicing packing solutions among n modules. We first enumerate all slicing tree structures and then enumerate all permutations of the modules. Let $L(n)$ be the number of different slicing tree structures for n modules. So we have

$$L(n) = \sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} L(n-i) \cdot L(i). \tag{10}$$

All slicing tree structures for 3–6 modules are listed in Fig. 8. Note that we are using the generalized slicing tree which does not differentiate the left–right order between two child subtrees. As we can see the number of different slicing tree structures is actually very limited.

To completely explore all slicing packing solutions among n modules, for each slicing tree structure, different permutations of the modules should also be considered. For example in Fig. 8, in tree T_{4a} four modules A, B, C, and D can be mapped to leaves "1–2–3–4" by the order "A–B–C–D" or "A–C–B–D." Obviously these two orders derive two different layouts. However, again because the generalized slicing tree does not differentiate the left–right order between two child subtrees which share the same parent node, for example, orders "A–C–B–D" and "B–A–C–D" are exactly the same in T_{4a}. After pruning such redundancy, we have $\frac{4!}{2} = 12$ nonredundant permutations for mapping four modules to the four leaves in T_{4a}. Therefore, for each slicing tree structure of n modules, we first enumerate all nonredundant permutations, for each one of which a shape curve is produced. And then we merge these curves into one curve associated with each

TABLE I
COMPARISON ON # OF "⊕" OPERATION

n	# of ⊕ by Naive Approach	# of ⊕ With DP
2	1	1
3	6	6
4	45	25
5	400	90
6	4155	301
7	49 686	966
8	674 877	3025
9	10 295 316	9330
10	174 729 015	28 501

slicing tree structure. Finally, these curves from all slicing tree structures are merged into one curve that captures all possible slicing layouts among these n modules. To show the amount of computations in this process, we list the number of "⊕" operations for different numbers of modules in the second column of Table I.

B. Enumeration by Dynamic Programming

Table I shows that the naive approach can be very expensive in both runtime and memory usage. Alternatively, we notice that the shape curve for a set of modules (M) can be defined recursively as follows:

$$S(M) = \underset{A \subset B, B = M - A}{\text{MERGE}} (S(A) \oplus S(B)). \tag{11}$$

$S(M)$ is the shape curve capturing all slicing layouts among modules in M, MERGE() is similar to the *Merging* in Fig. 5(c), but operates on shape curves from different sets.

Based on (11), we can use dynamical programming (DP) to implement the shape curve generation. First of all, we generate the shape curve representing the outline(s) of each module. For hard modules, there are two points[7] in each curve. For soft modules, only several points from each original curve are evenly sampled.[8] And then starting from the smallest subset of modules, we proceed to build up the shape curves for the larger subsets step by step, until the shape curve $S(M)$ is generated. Since in this process the previously generated curves can be reused for building up the curves of larger subsets of modules, many redundant computations are eliminated. After applying DP, the resulted numbers of "⊕" operations are listed in the third column of Table I.

C. Impact of EP on Packing

To control the quality of packing in EP, we can adjust the number of modules in the set. Consequently the impact on packing is: *The more modules a set contains, the more different slicing tree structures we explore, the more slicing layout possibilities we have, and thus the better quality of packing we will gain at the top level.*

[7]One point if the hard module is a square.

[8]The number of sampled points on the whole curve is determined by $\lfloor \frac{A_i}{A_0} \rho \rfloor + 4$, where A_i is the area of soft block i, A_0 is the total block area, and ρ is a constant ($\rho = 10\,000$ by default).

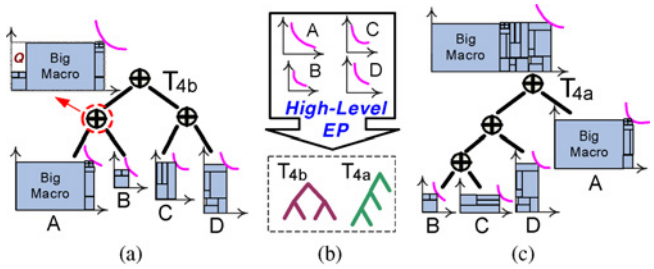

Fig. 9. Illustration of high-level EP.

However, if the set contains too many modules, two problems appear in EP: 1) the memory to store results from subsets can be expensive; and 2) since the interconnections among the modules are not considered, the wirelength may be increased. Due to these two concerns, in the first step of *DeFer*, we apply *hMetis* to recursively cut the original circuit into multiple smaller subcircuits. This process not only helps us to cut down the number of modules in each subcircuit, but initially optimizes the wirelength as well. Later on as applying EP on each subcircuit, the wirelength would not become a big concern, because this is only a locally packing exploration among a small number of modules. In other words, in the spirit of DDM, instead of deferring the decision on the slicing tree structure among all modules in the original circuit, first we fix the high-level slicing tree structure among the subcircuits by partitioning, and then defer the decision on the slicing tree structure among the modules within each subcircuit.

D. High-Level EP

In the modern system-on-a-chip design, the usage of *intellectual property* becomes more and more popular. As a result, a circuit usually contains numbers of big hard macros. Due to the big size differences from other small modules, they may produce some large whitespace. For example in Fig. 9(a), after partitioning, the original circuit has been cut into four subcircuits A, B, C, and D. A contains a big hard macro. Respecting the slicing tree structure of T_{4b}, you may find that no matter how hard EP explores various packing layouts within A or B, there is always a large whitespace, such as Q, in the parent subfloorplan. This is because the high-level slicing tree structure among subcircuits has been fixed by partitioning, so that some small subcircuit is forced to combine with some big subcircuit. Thus, to solve this problem, we need to explore other slicing tree structures among the subcircuits.

To do so, we apply EP on a set of *subfloorplans*, instead of a set of *modules*. As the input of EP is actually a set of shape curves, and shape curves can represent the shape of both subfloorplans and modules, it is capable of using EP to explore the layouts among subfloorplans. In Fig. 9(b), EP is applied on the four shape curves coming from subfloorplans A, B, C, and D, respectively. So all slicing tree structures (T_{4a} and T_{4b}) and permutations among these subfloorplans can be completely explored. Eventually one tightly-packed layout can be chosen during back-tracing step [see Fig. 9(c)].

Before we describe the criteria of triggering high-level EP, some concepts are introduced here as follows.

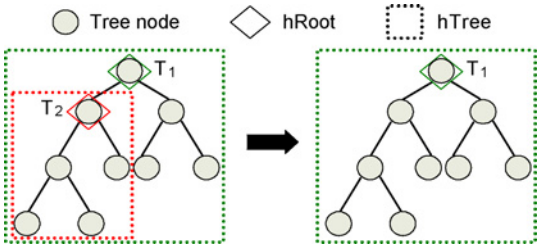

Fig. 10. One exception of identifying *hTree*.

- *Big gap:* Based on the definition of ΔH_p in Section IV, if $h^i_{p+1} - h^i_p > \omega \cdot \Delta H_p$ (ω is "Gap Ratio," $\omega = 5$ by default), then we say there is a "big gap" between points p and $p+1$. Intuitively, if there is a big gap, most likely it would cause serious packing problem at upper level.
- *hNode:* In the high-level slicing tree, the tree node or leaf node that contains big gap(s).
- *hTree:* A subtree of the high-level slicing tree, where the high-level EP is applied. For example, T_{4b} is a *hTree* [see Fig. 9(a)].
- *hRoot:* The root node of *hTree*.

High-level EP is to solve the packing problem caused by big gaps, so we need to identify the *hTree* that contains big gap. First we search for the big gap through the high-level slicing tree. If any shape curve has a big gap, then the corresponding node becomes an *hNode*. After identifying all *hNodes*, each *hNode* becomes an *hRoot*, and the subtree whose root node is *hRoot* becomes an *hTree*. But there is one exception: as shown in Fig. 10, if one *hTree* T_2 is a subtree of another *hTree* T_1, then T_2 will not become an *hTree*. Eventually, each *hTree* contains at least one big gap, which implies critical packing problems. Thus, for every *hTree* we use high-level EP to further explore the various packing layouts among the subfloorplans, i.e., leaves of *hTree*. If an *hTree* has more than 10 leaves, we will combine them from bottom-up until the number of leaves becomes 10.

As mentioned in Section V-C, EP only solves the packing issue, which may degrade the wirelength. Therefore, to make a trade-off we apply high-level EP only if there is no point enclosed into the fixed outline after combining step. If that is the case, then we will use the above criteria to trigger the high-level EP, and reconstruct the final shape curve.

VI. Block Swapping and Mirroring

After back-tracing step, the decision on subfloorplan order (left–right/top–bottom) has not been made yet. Using such property, this section focuses on optimizing the wirelength.

In slicing structures switching the order (left–right/top–bottom) of two child subfloorplans would not change the dimension of their parent floorplan outline, but it may actually improve the wirelength. Basically, we adopt three techniques here: 1) *Rough Swapping*; 2) *Detailed Swapping*; and 3) *Mirroring*. Each of them is trying to switch the positions of two subfloorplans to improve the half-perimeter wirelength (HPWL). Fig. 11 illustrates the differences between *Swapping* and *Mirroring*. In *Swapping*, we try to switch the left and

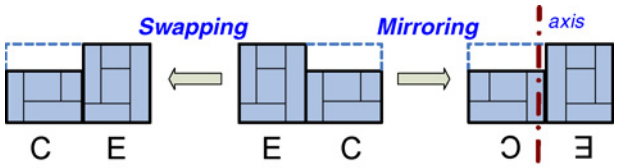

Fig. 11. *Swapping* and *Mirroring*.

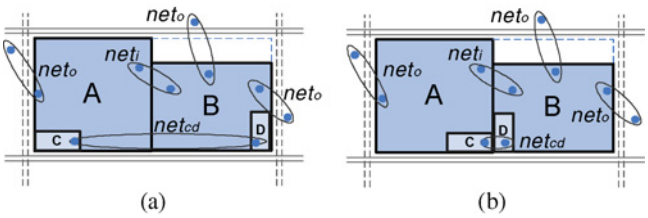

Fig. 12. Motivation on *Rough Swapping*.

right subfloorplans, inside of which the relative positions among the modules are unchanged. In *Mirroring*, instead of simply swapping two subfloorplans, we first figure out the symmetrical axis of the outline at their parent floorplan, and then attempt to mirror them based on this axis. When calculating the HPWL, in *Rough Swapping* we treat all internal modules to be at the center of their subfloorplan outline. In *Detailed Swapping*, we use the actual center coordinates of each module in calculating the HPWL.

Rough Swapping is an essential step before *Detailed Swapping*. Without it, the results produced by *Detailed Swapping* could degrade the wirelength. For example in Fig. 12, when we try to swap two subfloorplans A and B, two types of nets need to be considered: internal nets net_i between A and B, and external nets net_o between the modules inside A or B and other outside modules or fixed pads. Let C and D be two modules inside A and B, respectively. C and D are highly connected by net_{cd}. After back-tracing step, the coordinates of C and D are still unknown. If we randomly specify the positions of C and D as shown in Fig. 12(a), then we may swap A and B to gain better wirelength. Alternatively, if C and D are specified in the positions in Fig. 12(b), then we may not swap them. As we can see, the randomly specified module position may mislead us to make the wrong decision. To avoid such "noise" generated by net_i in the swapping process, the best thing to do is to assume C, D and all modules inside subfloorplans A and B are at the centers of A and B, such that the right decision can be made based on net_o.

Essentially, we first apply *Rough Swapping* from top-down, followed by *Detailed Swapping*. Finally, *Mirroring* is used. Note that the order between *Detailed Swapping* and *Mirroring* can be changed, and both of them can be applied from either top-down or bottom-up.

VII. Extension of DeFer

This section presents the different strategies of selecting the points from the final shape curve, such that *DeFer* is capable of handling floorplanning problems with various objectives.

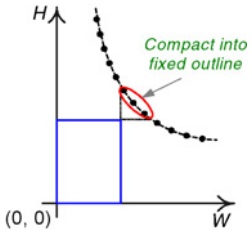

Fig. 13. Compact invalid points into fixed outline.

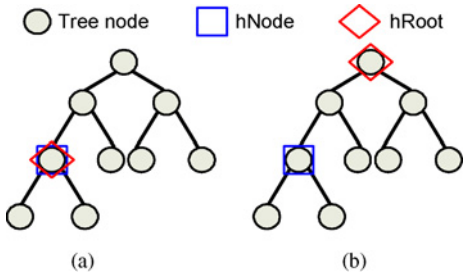

Fig. 14. Two strategies of identifying *hRoot*.

S : hMetis Initial Seed, GR : Gap Ratio, HS : hRoot Strategy, W : Weight Setting Method
****** Quit any run, once satisfy fixed-outline constraint ******
Run 1: hMetis(S), $GR = 5$, $HS = (2)$, $W = (1)$
Run 2: hMetis(S++), $GR = 5$, $HS = (2)$, $W = (1)$
Run 3: $GR = 5$, $HS = (1)$
Run 4: $GR = 4$, $HS = (1)$
Run 5: $GR = 3$, $HS = (1)$
Run 6: hMetis(S++), $GR = 3$, $HS = (1)$, $W = (2)$
Run 7: hMetis(S++), $GR = 3$, $HS = (1)$, $W = (2)$
Run 8: hMetis(S++), $GR = 3$, $HS = (1)$, $W = (2)$

Fig. 15. Tuned parameters at each run.

1) *Fixed-Outline Floorplanning:* Given the final shape curve, it is very straightforward to select the valid points enclosed into the fixed outline. Let P be the number of such valid points. As for each selected point the swapping process is applied to optimize the HPWL, to make a trade-off between runtime and solution quality *DeFer* chooses at most δ points ($\delta = 21$ by default) for the back-tracing. So we have three cases.
   a) $P > \delta$*:* Based on the geometric observation between aspect ratio and HPWL in [9], *DeFer* chooses δ points where the outline aspect ratio is closed to 1.
   b) $0 < P \leq \delta$*:* All P points are chosen.
   c) $P = 0$*: DeFer* still chooses at most δ points near the upper-right corner of the fixed outline (see Fig. 13), in that we attempt to compact them into the fixed outline in compacting step.
2) *Min-Area Floorplanning:* For min-area floorplanning, *DeFer* just needs to go through each points on the final shape curve and find out the one with the minimum area. Because the area minimization is the only objective here, we can even skip swapping step and shifting step to gain fast runtime. This problem considers to be very easy for *DeFer*.
3) *Min-Area and Wirelength Floorplanning:* This problem uses a linear combination of area and wirelength as the cost function. Compared with the strategy of fixed-outline floorplanning, the only difference is that we just need to choose the δ points with the minimum area, rather than within the fixed outline.

As shown above, *DeFer* is very easy to be switched to handle other floorplanning problems. Because once the final shape curve is available, *DeFer* has provided a large amount of floorplan candidates. Given *any objective function*, e.g., that used in simulated annealing, we just need to evaluate the candidates, and pick the one that gives the minimum cost.

VIII. Implementation Details

Sometimes *DeFer* cannot pack all modules into the fixed outline. This may occur because *hMetis* generates a hard-to-pack partition result, or the packing strength is not strong enough. To enhance the robustness of *DeFer*, we adaptively tune some parameters and try another run.

One effective way to improve the packing quality of *DeFer* is to enhance the packing strength in the high-level EP, e.g., by decreasing the gap ratio ω. Also, we can use different strategies to identify *hRoot* (see Fig. 14).

1) Each *hNode* becomes an *hRoot*.
2) Each *hNode*'s grandparent tree node becomes an *hRoot*.

Strategy 1 is the one we mentioned in Section V-D. Apparently, if we adopt strategy 1, more *hTrees* will be generated, and thus the high-level EP is used more often, which leads better packing. However, this takes longer runtime.

Another way to improve the packing quality is to balance both the area and number of modules, rather than only the area in each partition at partitioning step. Thus, we have two methods to set the weight for the module.

1) $Wgt = A_m$.
2) $Wgt = A_m + 0.6 \cdot \overline{A_p}$.

Here, Wgt and A_m are the weight and area for module m, $\overline{A_p}$ is the average module area in partition p. In experiments, we observe that method 2, which considers both the area and number of modules, generates better packing results, yet sacrifices the wirelength.

Essentially, *DeFer* starts with the defaulted parameters for the first run. If failing to pack all modules into the fixed outline, it will internally enhance the packing strength and try another run. By default *DeFer* will try at most eight runs. The tuned parameters for each run is listed in Fig. 15. For Runs 3–5, because they share the same partition result with Run 2, *DeFer* skips the partitioning step in those runs.

Even though *DeFer* internally executes multiple runs, it still achieves the best runtime compared with all other floorplanners. There are two reasons: 1) *DeFer* is so fast. Even it runs multiple times, it is still much faster than other floorplanners; and 2) *DeFer* has better packing quality. For most circuits, *DeFer* can satisfy the fixed-outline constraint within Run 1.

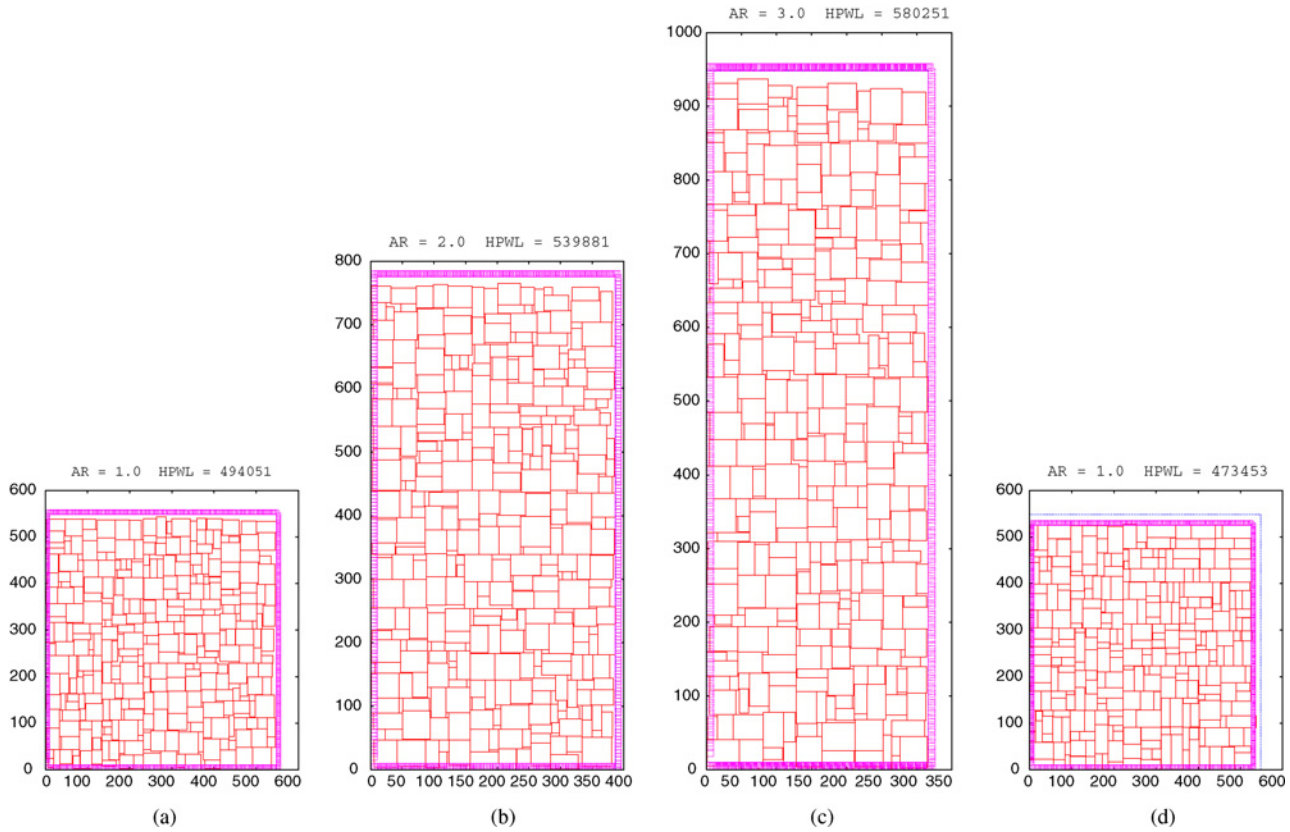

Fig. 16. Circuit $n300$ layouts generated by *DeFer*. (a) $n300$ hard block $\gamma = 10\%$. (b) $n300$ hard block $\gamma = 10\%$. (c) $n300$ hard block $\gamma = 10\%$. (d) $n300$ soft block $\gamma = 1\%$.

TABLE II

COMPARISON ON GSRC HARD-BLOCK BENCHMARKS [22] ($\gamma = 10\%$)

Circuit		$n100$			$n200$			$n300$			**Normallized**
Aspect Ratio		1	2	3	1	2	3	1	2	3	
Suc%	*Parquet 4.5*	42%	43%	33%	26%	19%	17%	16%	16%	14%	**0.25**
	FSA	100%	0%	0%	100%	0%	0%	0%	0%	0%	**0.22**
	IMF	100%	100%	100%	100%	100%	100%	100%	100%	100%	**1.00**
	IARFP	99%	100%	99%	100%	99%	63%	100%	100%	46%	**0.90**
	PATOMA	0%	0%	0%	0%	100%	0%	100%	100%	100%	**0.44**
	Capo 10.5	17%	17%	15%	0%	0%	2%	0%	1%	0%	**0.06**
	DeFer	100%	100%	100%	100%	100%	100%	100%	100%	100%	**1**
HPWL	*Parquet 4.5*	248 652	269 191	289 963	467 627	506 946	544 621	686 588	725 833	781 556	**1.27**
	FSA	243 823	–	–	414 777	–	–	–	–	–	**1.14**
	IMF	250 680	251 418	257 935	438 467	454 231	482 651	584 578	617 510	666 245	**1.14**
	IARFP	220 269	230 553	247 283	386 537	409 208	433 631	535 850	567 496	600 438	**1.03**
	PATOMA	–	–	–	–	483 110	–	653 711	697 740	680 671	**1.25**
	Capo 10.5	227 046	241 789	261 334	–	–	444 079	–	566 998	–	**1.05**
	DeFer	208 650	229 603	248 567	372 546	402 155	431 552	498 909	538 515	577 209	**1**
Time (s)	*Parquet 4.5*	10.85	10.58	10.27	44.43	44.47	41.96	95.02	87.03	86.31	**181.49**
	FSA	39.78	–	–	202.13	–	–	–	–	–	**557.74**
	IMF	7.65	10.82	9.29	41.21	43.59	38.71	74.74	71.48	71.72	**157.91**
	IARFP	4.44	4.50	4.52	16.51	15.48	14.22	29.30	29.48	30.03	**64.33**
	PATOMA	–	–	–	–	0.25	–	0.36	0.34	0.48	**1.15**
	Capo 10.5	122.64	125.18	160.07	–	–	3054	–	8661	–	**222.39**
	DeFer	0.13	0.11	0.11	0.25	0.23	0.22	0.35	0.33	0.33	**1**
#Valid Point/#Total Point		3/617	4/621	3/621	3/670	2/672	2/672	6/869	5/869	4/869	

TABLE III
COMPARISON ON GSRC SOFT-BLOCK BENCHMARKS [22] ($\gamma = 1\%$)

Circuit		$n100$			$n200$			$n300$			**Normallized**
Aspect Ratio		1	2	3	1	2	3	1	2	3	
Suc%	*Parquet 4.5*	0%	0%	0%	0%	0%	0%	0%	0%	0%	**0**
	Capo 10.5	0%	0%	0%	0%	0%	0%	0%	0%	0%	**0**
	PATOMA	100%	100%	100%	100%	100%	100%	100%	100%	100%	**1.00**
	DeFer	100%	100%	100%	100%	100%	100%	100%	100%	100%	**1**
HPWL	*Parquet 4.5*	–	–	–	–	–	–	–	–	–	**—**
	Capo 10.5	–	–	–	–	–	–	–	–	–	**—**
	PATOMA	215 455	213 561	230 759	383 330	367 565	404 574	524 774	486 351	518 204	**1.01**
	DeFer	196 457	217 686	235 702	354 885	380 470	410 464	476 508	514 764	551 610	**1**
Time (s)	*Parquet 4.5*	–	–	–	–	–	–	–	–	–	**—**
	Capo 10.5	–	–	–	–	–	–	–	–	–	**—**
	PATOMA	0.39	0.40	0.38	0.92	0.93	0.83	1.28	1.28	1.37	**3.50**
	DeFer	0.09	0.09	0.09	0.18	0.19	0.19	0.78	0.96	0.97	**1**
#Valid Point/#Total Point		28/20 392	30/20 469	30/20 469	16/25 513	18/25 493	17/25 493	9/30 613	10/30 598	10/30 603	

IX. EXPERIMENTAL RESULTS

In this section, we present the experimental results. All experiments were performed on a Linux machine with Intel Core Duo[9] 1.86 GHz CPU and 2 GB memory. The wirelength is measured by HPWL. We compare *DeFer* with all the best publicly available state-of-the-art floorplanners, of which the binaries are the latest version. For the *hMetis 1.5* parameters in *DeFer*, $NRuns = 1$, $UBfactor = 10$, and others are defaulted.

A. Experiments on Fixed-Outline Floorplanning

In this section, we compare *DeFer* with other fixed-outline floorplanners. On *GSRC* and *HB* benchmarks, for each circuit we choose three different fixed-outline aspect ratios: $\tau = 1, 2, 3$. All input/output (I/O) pads are scaled to the according boundary. On *HB+* benchmarks, we use the defaulted fixed outlines and I/O pad locations. By default every floorplanner runs 100 times for each test case, and the results are averaged over all *successful* runs. As *PATOMA* has internally fixed the *hMetis* seed, and produces the same result no matter how many times it runs, we run it only once. For other floorplanners, the initial seed is the same as the index of each run. *Parquet 4.5* runs in wirelength minimization mode. The parameters for other floorplanners are defaulted. For each type of benchmarks, we finally normalize all results to *DeFer's* results.

1) *GSRC Hard-Block Benchmarks:* These circuits contain 100, 200, and 300 hard modules. *DeFer* compares with six floorplanners: *Parquet 4.5* [4], *FSA* [6], *IMF* [8], *IARFP* [9], *PATOMA* [14], and *Capo 10.5* [5]. The maximum whitespace percentage $\gamma = 10\%$. The results are summarized in Table II. For every test case *DeFer* reaches 100% success rate. *DeFer* generates 27%, 14%, 14%, 3%, 25%, and 5% better HPWL in 181×, 558×, 158×, 64×, 15%, and 222× faster runtime than *Parquet 4.5*, *FSA*, *IMF*, *IARFP*, *PATOMA*, and *Capo 10.5*, respectively. *DeFer* consistently achieves the *best* HPWL and *best* runtime on all 9 test cases, except for only one case ($n100$, $\tau = 3$) *DeFer* generates 0.5% worse HPWL than *IARFP*. But for that one *DeFer* is 41× faster than *IARFP* with 100% success rate. Fig. 16(a)–(c) shows the layouts produced by *DeFer* on circuit $n300$ with $\tau = 1, 2, 3$.

[9]In the experiments, only one core was used.

2) *GSRC Soft-Block Benchmarks:* These circuits contain 100, 200, and 300 soft modules. *DeFer* compares with *Parquet 4.5*, *Capo 10.5*, and *PATOMA*, as only these floorplanners can handle soft modules. We add "-soft" to *Parquet 4.5* command line. The maximum whitespace percentage $\gamma = 1\%$, which is almost zero whitespace requirements. As we can see from Table III, after 100 runs both *Parquet 4.5* and *Capo 10.5* cannot pack all modules within the fixed outline. *PATOMA* and *DeFer* reach 100% success rate on every test case. Compared with *PATOMA*, *DeFer* generates 1% better wirelength with 4× faster runtime. Fig. 16(d) is the final layout generated by *DeFer* on circuit $n300$ with $\tau = 1$, which shows almost 0% whitespace is reached.

3) *HB Benchmarks:* We compare *DeFer* with *PATOMA* and *Capo 10.5* on *HB* benchmarks. These circuits are generated from the IBM/ISPD98 suite containing both hard and soft modules ranging from 500 to 2000, some of which are big hard macros. Detailed statistics are listed in the second column of Table IV. To get better runtime, wirelength and success rate, we run *Capo 10.5* in "-SCAMPI" [23] mode. However, *Capo 10.5* still takes a long time to finish one run for each test case, so we only run it *once* with the defaulted seed. To show its slowness, we also list the reported runtime for the *unsuccessful* runs. From Table IV, we can see that *DeFer* does not achieve 100% success rate for only one test case, and the success rate is 2.33× and 8.33× higher than *PATOMA* and *Capo 10.5*. *Capo 10.5* crashes on four test cases, and takes more than two days to finish one test case. Compared with *PATOMA*, *DeFer* is 28% better on average in HPWL, and 3× faster. Compared with *Capo 10.5*, *DeFer* generates as much as 72% better HPWL with even 790× faster runtime. We also run *Parquet 4.5* on these circuits. However, it is so slow that even running one test case *once* takes thousands of seconds. So for each test case, we only run it once instead of 100 times, but none of the results fits into the fixed outline. Fig. 17(a)–(c) shows the layouts generated by *PATOMA*, *Capo 10.5*, and *DeFer* on circuit ibm03 with $\tau = 2$.

4) *HB+ Benchmarks: DeFer* compares with *PATOMA* and *Capo 10.5* on *HB+* benchmarks. These circuits are generated from *HB* benchmarks, while the biggest hard macro is inflated

TABLE IV
COMPARISON ON HB BENCHMARKS [24] (γ = 10%)

Circuit	#Soft./#Hard. /#Net.	Aspect Ratio	*PATOMA* [14] Suc%	*PATOMA* [14] WL (e+06)	*PATOMA* [14] Time (s)	*Capo 10.5* [5] Suc%	*Capo 10.5* [5] WL (e+06)	*Capo 10.5* [5] Time (s)	*DeFer* Suc%	*DeFer* WL (e+06)	*DeFer* Time (s)	#Valid Point /#Total Point
ibm01	665 /246 /4236	1 2 3	100% 0% 100%	2.84 – 5.60	7.04 – 1.66	0% 0% 0%	– – –	183 977 696	100% 100% 100%	2.66 2.70 2.82	1.44 1.28 1.30	16/1571 11/1482 12/1490
ibm02	1200 /271 /7652	1 2 3	0% 0% 0%	– – –	– – –	0% – 0%	– – –	456 *> 2 days* 3726	85% 100% 100%	6.55 6.21 6.29	14.48 3.33 3.52	6/2348 7/1161 10/1144
ibm03	999 /290 /7956	1 2 3	100% 100% 0%	12.59 12.94 –	5.42 5.58 –	100% 100% 0%	10.70 12.01 –	566 1874 2028	100% 100% 100%	8.77 8.89 8.99	3.60 3.49 3.59	59/2684 40/2503 44/2630
ibm04	1289 /295 /10 055	1 2 3	0% 0% 0%	– – –	– – –	0% 100% 100%	– 17.77 16.32	2752 5253 2262	100% 100% 100%	8.94 8.96 9.64	3.04 3.12 6.31	4/1492 9/1514 12/2685
ibm05	564 /0 /7887	1 2 3	100% 100% 100%	12.27 12.60 13.19	14.21 13.68 13.85	0% 0% 0%	– – –	458 358 411	100% 100% 100%	12.61 12.73 13.45	3.55 3.52 3.53	46/3369 46/3371 46/3371
ibm06	571 /178 /7211	1 2 3	0% 0% 0%	– – –	– – —	0% 0% 0%	– – –	235 592 2831	100% 100% 100%	7.87 7.76 8.91	3.66 3.66 3.60	53/2187 41/2235 36/2196
ibm07	829 /291 /11 109	1 2 3	0% 100% 100%	– 24.64 24.34	– 7.85 8.68	0% 0% 0%	– – –	1094 1270 2274	100% 100% 100%	13.81 13.91 14.32	3.87 4.48 4.26	12/1527 22/1625 18/1590
ibm08	968 /301 /11 536	1 2 3	0% 0% 0%	– – –	– – –	0% 0% 0%	– – –	2527 1110 1958	100% 100% 100%	13.95 14.16 14.43	5.44 5.40 5.55	15/1333 17/1290 19/1309
ibm09	860 /253 /11 008	1 2 3	0% 0% 0%	– – –	– – –	0% 0% 100%	– – 34.48	2273 2670 6652	100% 100% 100%	12.85 12.57 12.98	2.60 3.77 3.54	3/1495 17/1486 14/1486
ibm10	809 /786 /16 334	1 2 3	100% 0% 0%	48.47 – –	21.71 – –	0% *Crashed* 100%	– *Crashed* 53.64	2353 *Crashed* 2014	100% 100% 100%	33.25 34.23 36.59	11.63 18.00 16.52	9/2576 14/2897 9/2725
ibm11	1124 /373 /16 985	1 2 3	100% 0% 0%	20.87 – –	33.87 – –	0% 0% 0%	– – –	8070 4732 2245	100% 100% 100%	21.99 22.13 22.83	4.84 4.96 4.67	12/2218 8/2207 7/2174
ibm12	582 /651 /11 873	1 2 3	0% 0% 0%	– – –	– – –	0% 0% 0%	– – –	3085 864 19 952	100% 100% 100%	29.72 31.53 32.16	10.95 7.71 4.59	20/2909 18/3011 8/1957
ibm13	530 /424 /14 202	1 2 3	0% 100% 0%	– 43.81 –	– 9.84 –	0% 0% 0%	– – –	3401 3662 3201	100% 100% 100%	25.92 25.46 26.47	6.03 3.79 3.83	12/2553 10/2048 8/2095
ibm14	1021 /614 /26 675	1 2 3	100% 100% 100%	71.87 55.99 61.65	23.59 35.65 35.12	0% 0% 0%	– – –	4253 10 373 4976	100% 100% 100%	50.83 51.67 53.71	9.69 9.70 9.70	30/2976 34/2971 36/2971
ibm15	1019 /393 /28 270	1 2 3	0% 0% 0%	– – –	– – –	0% 0% 0%	– – –	3634 6827 2902	100% 100% 100%	64.18 63.17 66.06	9.71 9.13 9.46	25/1651 19/1580 20/1623
ibm16	633 /458 /21 013	1 2 3	0% 100% 100%	– 88.33 98.77	– 16.55 22.94	*Crashed* 0% 0%	*Crashed* – –	*Crashed* 8928 11 675	100% 100% 100%	56.88 58.55 59.91	16.79 14.55 12.84	18/3823 24/4833 18/4093
ibm17	682 /760 /30 556	1 2 3	100% 100% 100%	102.45 96.46 98.18	41.75 46.63 42.45	*Crashed* 0% *Crashed*	*Crashed* – *Crashed*	*Crashed* 2250 *Crashed*	100% 100% 100%	95.92 95.48 100.82	10.43 10.41 10.42	32/3253 27/3252 29/3252
ibm18	658 /285 /21 191	1 2 3	100% 100% 100%	50.28 49.74 52.26	38.24 39.15 36.97	0% 0% 0%	– – –	1083 4630 5262	100% 100% 100%	49.12 49.29 51.39	7.93 7.97 7.97	42/3106 41/3128 41/3128
Normalized			**0.43**	**1.28**	**3.28**	**0.12**	**1.72**	**789.79**	**1**	**1**	**1**	

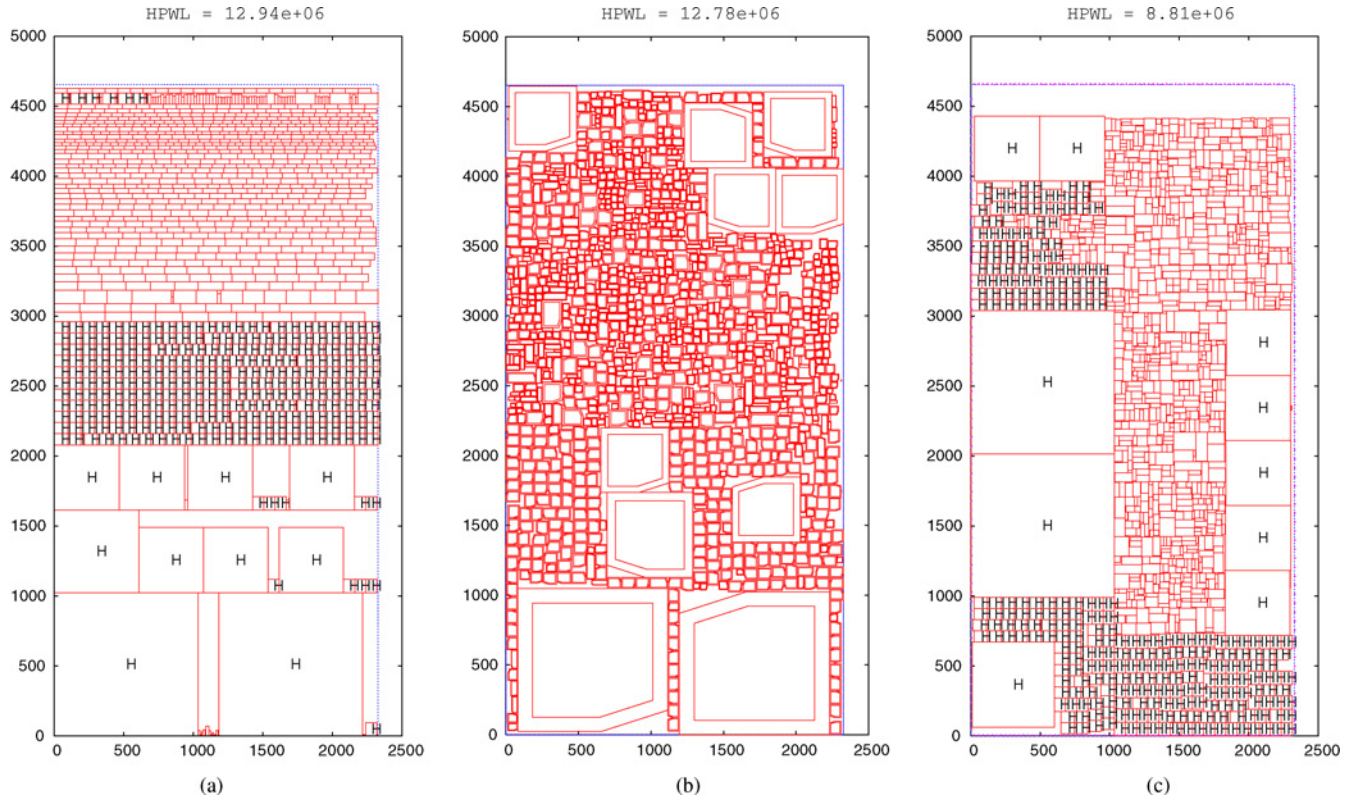

Fig. 17. Circuit ibm03 layouts generated by *PATOMA*, *Capo 10.5*, and *DeFer* ($\gamma = 10\%$ and $\tau = 2$). (a) By *PATOMA*. (b) By *Capo 10.5*. (c) By *DeFer*.

TABLE V
COMPARISON ON HB+ BENCHMARKS [23]

Circuit	White-space γ	Aspect Ratio	*PATOMA* [14]			*Capo 10.5* [5]			*DeFer*			#Valid Point /#Total Point
			Suc%	WL (e+06)	Time (s)	Suc%	WL (e+06)	Time (s)	Suc%	WL (e+06)	Time (s)	
ibm01	26%	1	100%	4.67	4.44	–	–	*> 2 days*	100%	3.09	1.84	120/10 860
ibm02	25%	1	0%	–	–	100%	7.86	124	100%	6.17	15.28	45/3380
ibm03	30%	1	0%	–	–	100%	12.75	343	100%	9.19	4.01	102/5020
ibm04	25%	1	0%	–	–	100%	12.03	147	100%	10.26	14.15	63/5170
ibm06	25%	1	0%	–	–	100%	10.09	155	100%	8.78	5.01	84/3560
ibm07	25%	1	100%	16.38	23.41	100%	16.41	99	100%	15.48	4.55	12/3780
ibm08	26%	1	0%	–	–	100%	18.29	284	100%	18.73	19.25	106/5070
ibm09	25%	1	100%	16.62	25.45	100%	17.85	100	100%	16.66	4.22	12/3070
ibm10	20%	1	0%	–	–	100%	81.27	1685	100%	45.12	6.32	27/6880
ibm11	25%	1	100%	25.86	38.72	100%	28.26	149	100%	26.99	7.07	19/4150
ibm12	26%	1	0%	–	–	100%	52.46	126	100%	50.17	5.54	69/6880
ibm13	25%	1	100%	36.74	29.08	100%	40.22	299	100%	35.51	5.85	15/3860
ibm14	25%	1	100%	68.30	51.79	100%	73.89	410	100%	64.50	12.01	36/7870
ibm15	25%	1	0%	–	–	100%	92.79	474	100%	84.29	14.66	182/9900
ibm16	25%	1	100%	95.97	47.14	100%	153.02	595	100%	98.66	8.08	10/5770
ibm17	25%	1	100%	142.41	65.06	100%	146.03	440	100%	144.56	14.70	41/9540
ibm18	25%	1	100%	73.76	47.71	100%	75.92	224	100%	71.86	11.30	44/9160
Normalized			**0.53**	**1.07**	**4.76**	**1.00**	**1.19**	**46.66**	**1**	**1**	**1**	

TABLE VI
COMPARISON ON LINEAR COMBINATION OF HPWL AND AREA

Circuit	*Parquet 4.5* [4]					*DeFer*				
	Area	Whitespace%	HPWL	Area + HPWL	Time (s)	Area	Whitespace%	HPWL	Area + HPWL	Time (s)
$n100$	194 425	8.31%	235 070	429 495	13.66	191 164	6.50%	209 785	400 949	0.33
$n200$	191 191	8.82%	438 584	629 775	54.84	187 734	6.85%	374 676	562 410	0.74
$n300$	298 540	9.29%	628 422	926 962	108.70	291 385	6.67%	503 311	794 696	0.96
Normalized	**1.02**	**1.32**	**1.18**	**1.12**	**76.24**	**1**	**1**	**1**	**1**	**1**

TABLE VII
ESTIMATION ON CONTRIBUTIONS OF MAIN TECHNIQUES AND RUNTIME BREAKDOWN IN *DeFer*

Algorithm Step		**Partitioning**		**Combining**		**Back-tracing**	**Swapping**			**Compacting**	**Shifting**
Main Technique		*Min-Cut*	*TP*	*EP*	*Combination*	–	*Rough*	*Detailed*	*Mirroring*	*Compaction*	*Shifting*
Wirelength Improvement		**Major**	Minor	–	–	–	**Major**	Minor	Minor	Minor	Minor
Packing Improvement		Minor	–	**Major**	Minor	–	–	–	–	Minor	—
Runtime Breakdown	GSRC Hard	29%		63%		0%	8%			0%	0%
	GSRC Soft	35%		37%		0%	28%			0%	0%
	HB	52%		4%		0%	44%			0%	0%
	HB+	46%		3%		0%	51%			0%	0%

by 100% and the area of remaining soft macros are reduced to preserve the total cell area. As a result, the circuits become even harder to handle. Due to the same reason, we set *Capo 10.5* to "-SCAMPI" mode, and run it only once. The results are shown in Table V. *DeFer* achieves the 100% success rate on all circuits, which is 1.89× better than *PATOMA*. *Capo 10.5* also achieves 100% success rate, expect for one circuit it takes more than two days to finish. In terms of the HPWL comparison, *DeFer* is 7% and 19% better than *PATOMA* and *Capo 10.5*. *DeFer* is also 5× and 47× faster than *PATOMA* and *Capo 10.5*.

Both *HB* and *HB+* benchmarks are considered to be very hard to handle, because these circuits not only contain both hard and soft modules, but also big hard macros. As far as we know, only the above floorplanners can handle these circuits. Obviously, *DeFer* reaches the *best* result. We also monitor the memory usage of *DeFer* on such large-scale circuits, and observe that the peak memory usage in *DeFer* is only 53 MB.

5) *Analysis of Points in* DeFer*:* In Tables II–V, for each test case we list the number of valid points (#VP) within the fixed outline and the total number of points (#FP) in the final shape curve. Both #VP and #FP are averaged over all successful runs. We have three observations as follows.

- As the circuit size grows, #FP increases.
- For the same circuit with various τ, ideally #FP should be the same. But they are actually different in some test cases. It is because high-level EP reconstructed the final shape curve for some hard-to-pack instances. As you can see high-level EP can significantly increase #FP, e.g., *ibm12* in Table IV, which means it improves packing quite effectively.
- Sometimes while other algorithms cannot satisfy the fixed-outline constraint, #VP of *DeFer* is more than 100, e.g., *ibm15* in Table V. This shows *DeFer*'s superior packing ability.

B. Experiments on Classical Outline-Free Floorplanning

For the classical outline-free floorplanning problem, as far as we know, only *Parquet 4.5* can handle *GSRC* benchmarks, so we compare it with *DeFer* on *GSRC Hard-Block* benchmarks. The results are averaged over 100 runs. The objective function is a linear combination of the HPWL and area, which are equally weighted. We add "-minWL" to the *Parquet 4.5* command line. As shown in Table VI, *DeFer* produces 32% less whitespace than *Parquet 4.5*, with 18% less wirelength. Overall, *DeFer* is 12% better in the total cost, and 76× faster than *Parquet 4.5*.

X. CONCLUSION

As the earliest stage of VLSI physical design, floorplanning has numerous impacts on the final performance of ICs. In this paper, we have proposed a fast, high-quality, scalable and nonstochastic fixed-outline floorplanner *DeFer*.

Based on the principle of *Deferred Decision Making*, *DeFer* outperforms all other state-of-the-art floorplanners in every aspect. It is hard to accurately calculate how much each technique in *DeFer* contributes to the overall significant improvement. But we do have a rough estimation in Table VII, in which we also show the runtime breakdown of *DeFer* for each set of benchmarks. Note that, the DDM idea is the soul of *DeFer*. Without it, those techniques cannot be integrated in such a nice manner and produce promising results.

Such a high-quality and efficient floorplanner is expected to handle the increasing complexity of modern designs. The source code of *DeFer* and all benchmarks are publicly available at [25]. In the future, we will integrate *DeFer* into placement tools to handle large-scale mixed-size designs.

ACKNOWLEDGMENT

The authors would like to thank Dr. I. Markov, S. Chen, T.-C. Chen, and the University of California, Los Angeles Computer-Aided Design group, for their help with *Capo 10.5*, *IARFP*, *IMF*, and *PATOMA*, respectively. They are also grateful to Dr. T.-C. Wang and the anonymous reviewers for their helpful suggestions and comments on this paper.

REFERENCES

[1] A. B. Kahng, "Classical floorplanning harmful?," in *Proc. Int. Symp. Phys. Design*, 2000, pp. 207–213.
[2] R. H. J. M. Otten, "Efficient floorplan optimization," in *Proc. Int. Conf. Comput. Design*, 1983, pp. 499–502.
[3] S. N. Adya and I. L. Markov, "Fixed-outline floorplanning through better local search," in *Proc. Int. Conf. Comput. Design*, 2001, pp. 328–334.
[4] S. N. Adya and I. L. Markov, "Fixed-outline floorplanning: Enabling hierarchical design," *IEEE Trans. Very Large Scale Integrat. Syst.*, vol. 11, no. 6, pp. 1120–1135, Dec. 2003.
[5] J. A. Roy, S. N. Adya, D. A. Papa, and I. L. Markov, "Min-cut floorplacement," *IEEE Trans. Comput.-Aided Design Integrat. Circuits Syst.*, vol. 25, no. 7, pp. 1313–1326, Jul. 2006.
[6] T.-C. Chen and Y.-W. Chang, "Modern floorplanning based on B*-trees and fast simulated annealing," *IEEE Trans. Comput.-Aided Design Integrat. Circuits Syst.*, vol. 25, no. 4, pp. 637–650, Apr. 2006.
[7] Y. C. Wang, Y. W. Chang, G. M. Wu, and S. W. Wu, "B*-Tree: A new representation for nonslicing floorplans," in *Proc. Design Automat. Conf.*, 2000, pp. 458–463.

[8] T.-C. Chen, Y.-W. Chang, and S.-C. Lin, "A new multilevel framework for large-scale interconnect-driven floorplanning," *IEEE Trans. Comput.-Aided Design Integrat. Circuits Syst.*, vol. 27, no. 2, pp. 286–294, Feb. 2008.

[9] S. Chen and T. Yoshimura, "Fixed-outline floorplanning: Enumerating block positions and a new objective function for calculating area costs," *IEEE Trans. Comput.-Aided Design Integrat. Circuits Syst.*, vol. 27, no. 5, pp. 858–871, May 2008.

[10] O. He, S. Dong, J. Bian, S. Goto, and C.-K. Cheng, "A novel fixed-outline floorplanner with zero deadspace for hierarchical design," in *Proc. Int. Conf. Comput. Aided Design*, 2008, pp. 16–23.

[11] A. Ranjan, K. Bazargan, S. Ogrenci, and M. Sarrafzadeh, "Fast floorplanning for effective prediction and construction," *IEEE Trans. Very Large Scale Integrat. Syst.*, vol. 9, no. 2, pp. 341–352, Apr. 2001.

[12] P. G. Sassone and S. K. Lim, "A novel geometric algorithm for fast wire-optimized floorplanning," in *Proc. Int. Conf. Comput. Aided Design*, 2003, pp. 74–80.

[13] Y. Zhan, Y. Feng, and S. S. Sapatnekar, "A fixed-die floorplanning algorithm using an analytical approach," in *Proc. Asia South Pacific-Design Automat. Conf.*, 2006, pp. 771–776.

[14] J. Cong, M. Romesis, and J. R. Shinnerl, "Fast floorplanning by look-ahead enabled recursive bipartitioning," *IEEE Trans. Comput.-Aided Design Integrat. Circuits Syst.*, vol. 25, no. 9, pp. 1719–1732, Sep. 2006.

[15] J. Z. Yan and C. Chu, "DeFer: Deferred decision making enabled fixed-outline floorplanner," in *Proc. Design Automat. Conf.*, 2008, pp. 161–166.

[16] M. Lai and D. F. Wong, "Slicing tree is a complete floorplan representation," in *Proc. Design, Automat. Test Eur.*, 2001, pp. 228–232.

[17] L. Stockmeyer, "Optimal orientations of cells in slicing floorplan designs," *Informat. Control*, vol. 57, pp. 91–101, May–Jun. 1983.

[18] G. Zimmerman, "A new area and shape function estimation technique for very large scale integration layouts," in *Proc. Design Automat. Conf.*, 1988, pp. 60–65.

[19] A. E. Dunlop and B. W. Kernighan, "A procedure for placement of standard-cell very large scale integration circuits," *IEEE Trans. Comput.-Aided Design Integrat. Circuits Syst.*, vol. 4, no. 1, pp. 92–98, Jan. 1985.

[20] G. Karypis and V. Kumar, "Multilevel k-way hypergraph partitioning," in *Proc. Design Automat. Conf.*, 1999, pp. 343–348.

[21] S. Goto, "An efficient algorithm for the 2-D placement problem in electrical circuit layout," *IEEE Trans. Circuits Syst.*, vol. 28, no. 1, pp. 12–18, Jan. 1981.

[22] GSRC Floorplan Benchmarks [Online]. Available: http://vlsicad.eecs.umich.edu/BK/GSRCbench/

[23] A. Ng, I. L. Markov, R. Aggarwai, and V. Ramachandran, "Solving hard instances of floorplacement," in *Proc. Int. Symp. Phys. Design*, 2006, pp. 170–177.

[24] HB Floorplan Benchmarks [Online]. Available: http://cadlab.cs.ucla.edu/cpmo/HBsuite.html

[25] DeFer Source Code [Online]. Available: http://www.public.iastate.edu/∼zijunyan/

Jackey Z. Yan received the B.S. degree in automation from the Huazhong University of Science and Technology, Wuhan, China, in 2006. He is currently pursing the Ph.D. degree in computer engineering at the Department of Electrical and Computer Engineering, Iowa State University, Ames.

His research interests include very large scale integration physical designs, specifically in algorithms for floorplanning and placement, and physical synthesis integrated system-on-a-chip designs.

Mr. Yan's work on fixed-outline floorplanning was nominated for the Best Paper Award at the Design Automation Conference in 2008. He received the Ultra-Excellent Student Award from Renesas Technology Corp., Tokyo, Japan, in 2005.

Chris Chu received the B.S. degree from the University of Hong Kong, Hong Kong, in 1993, and the M.S. and Ph.D. degrees from the University of Texas, Austin, in 1994 and 1999, respectively, all in computer science.

Since 1999, Chu has been a Faculty with Iowa State University, Ames. He is currently an Associate Professor with the Department of Electrical and Computer Engineering, Iowa State University. His research interests include computer-aided design of very large scale integration physical designs, and design and analysis of algorithms.

Dr. Chu received the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS Best Paper Award in 1999 for his work in performance-driven interconnect optimization. He received the International Symposium on Physical Design (ISPD) Best Paper Award in 2004 for his work in efficient placement algorithm. He received the Bert Kay Best Dissertation Award in 1998–1999 from the Department of Computer Sciences, University of Texas. He has served on the technical program committees of several major conferences including the Design Automation Conference, the International Conference on Computer-Aided Design, the ISPD, the International Symposium on Circuits and Systems, the Design, Automation and Test in Europe, the Asia and South Pacific Design Automation Conference, and the system level interconnect prediction.