Hindawi Publishing Corporation
VLSI Design
Volume 2011, Article ID 530851, 10 pages
doi:10.1155/2011/530851

Research Article

Buffer Planning for IP Placement Using Sliced-LFF

Ou He,[1] Sheqin Dong,[1] Jinian Bian,[1] and Satoshi Goto[2]

[1] *Tsinghua National Laboratory for Information Science & Technology, Tsinghua University, Beijing 100084, China*
[2] *Information, Production and Systems (IPS), Waseda University, Kitakyushu-shi 808-0135, Japan*

Correspondence should be addressed to Ou He, ho06@mails.tsinghua.edu.cn

Received 14 November 2010; Accepted 11 December 2010

Academic Editor: Shiyan Hu

IP cores are widely used in modern SOC designs. Hierarchical design has been employed for the growing design complexity, which stimulates the need for fixed-outline floorplanning. Meanwhile, buffer insertion is usually adopted to meet the timing requirement. In this paper, buffer insertion is considered with a fixed-outline constraint using Less Flexibility First (LFF) algorithm. Compared with Simulated Annealing (SA), our work is able to distinguish geometric differences between two floorplan candidates, even if they have the same topological structure. This is helpful to get a better result for buffer planning since buffer insertion is quite sensitive to a geometric change. We also extend the previous LFF to a more robust version called Sliced-LFF to improve buffer planning. Moreover, a 2-staged LFF framework and a post-greedy procedure are introduced based on our net-classing strategy and finally achieve a significant improvement on the success rate of buffer insertion (40.7% and 37.1% in different feature sizes). Moreover, our work is much faster than SA, since it is deterministic without iterations.

1. Introduction

IP cores, which are modeled as quantities of small, hard, and mixed-size macros [1], will dominate modern SOC designs. Hierarchical floorplanning then becomes essential. Different from traditional outline-free floorplanning, the fixed-outline constraint, which enables hierarchical design [2], will play a fundamental role in modern SOC designs.

Besides, in deep submicron technology, interconnect delay determines the chip performance. Buffer insertion is helpful to reduce the delay. However in SOC designs, buffers cannot be inserted into hard IP modules, because they consume silicon recourses as well, which will cause a redesign of hard IPs. As a result, they could only be inserted in the interspace between modules. So, it is better to consider buffer insertion together with IP module placement. In this paper, we integrate buffer planning into floorplanning stage, which places IP modules and reserves silicon space for buffers with a fixed-outline constraint.

Many previous works have been addressed in floorplanning stage considering buffer insertion. Cong et al. [3] first gave the concept of "Feasible Region" (FR) to calculate all the feasible spots to insert a buffer and meet the timing requirement of the net. Sarkar et al. [4] improved the notion "FR" into the "Independent Feasible Region" (IFR). IFR means a region where a buffer can be placed as long as the other buffers are inserted successfully in their IFRs. Using the net flow method, Tang and Wong [5] proposed an optimal algorithm to allocate buffers into buffer blocks with the assumption that every net only needed one buffer. Dragan et al. [6] put buffers into an existing block, which was implemented using the multicommodity flow-based theory. Sham et al. [7] released a routability-driven floorplanning system, which was able to estimate needs and resources of buffers with the consideration of routing congestions. Rafiq et al. [8] proposed a floorplanner for bus-based microprocessors with buffer and channel insertion. Alpert et al. [9] solved buffer planning problem using tile graph and dynamic programming. They assumed that buffers could be inserted into the IP modules, which was relied on the future design methodology of IP modules. Chen et al. [10] proposed a post-buffer planning algorithm based on deadspace redistribution. Deadspace means the interspace among the placed modules. Jiang et al. [11] implemented floorplanning and buffer block planning simultaneously. In each iteration of SA, they constructed a routing tree for each net, allocated buffers, and introduced buffer blocks into the intermediate floorplan. Ma et al. [12] solved buffer planning

as an integral part of floorplanning with the consideration of routing congestions. This work was also based on Simulated Annealing. Dong et al. [13] resized the circuit modules to insert more buffers. But this method is not practical for hard IP cores in SOC design, because their dimensions are always fixed and cannot be resized. Both buffer and interlayer via planning were considered by He et al. [14] for 3D ICs.

Most of the aforementioned works adopted either of two different strategies to do buffer planning. One is to take the problem as a postprocess after floorplanning. In this case, a seed floorplan is generated at first. Then, based on the existing floorplan, we reshape deadspace for buffers with a fixed topological structure, such as [6, 10]. However, all the works are finished in a given topological structure; so the solution space is limited. The other is to consider buffer planning with floorplanning. This methodology evaluates different topological structures and finally picks the best one to improve the number of inserted buffers, such as [12]. However, this method focuses on topological structures and fails to distribute deadspace effectively. More details will be discussed in Section 4.6. Besides, most of SA-based previous works did not consider the fixed outline constraint.

Section 4.6 explains the fact that buffer insertion issue is sensitive to geometric structures. That means that we should consider not only different topological structures but also the deadspace distribution with a fixed topology. Both of them will change the geometric structure of a floorplan. Unlike the other previous works, [11] could consider these two at the same time, but it was time-consuming using an SA framework.

In this paper, another algorithm framework named Less Flexibility First (LFF) [15] is adopted to handle buffer insertion issue. It works in a fixed-outline constraint unlike the aforementioned previous works. Compared with SA, LFF can handle buffer insertion and fixed-outline constraint together and run much faster. In LFF, different packing orders correspond to different topological structures. Thus, we can implement topological optimization by choosing the order of packing. In addition, we can also reshape the deadspace for buffers by inserting some fictitious modules in IP placement.

Moreover, LFF can work together with SA framework. Results of our LFF buffer planning can also be used to provide an initial solution for SA to improve the efficiency of SA iterations. Meanwhile, Half Perimeter Wirelength (HPWL) is still used to optimize timing in this paper. This is because most previous works also adopted wirelength (or weighted wirelength) for timing optimization. Moreover, HPWL is easy and clear to compare with previous works.

Detailed contributions in this paper are listed as follows.

(i) A new version for LFF called Sliced-LFF is introduced to represent the deadspace and enhance the robustness of the packing process. This work will facilitate buffer planning as well, since more information about the deadspace could be provided by our slice list for buffers.

(ii) Buffer insertion is integrated into LFF process. After that, topological and geometric information is both handled in our algorithm for buffer optimization.

(iii) Net-classing strategy is proposed for further optimization.

(iv) A post-floorplanning method is adopted to greedily insert more buffers.

Experimental results show that a significant improvement on the success rate of buffer insertion (40.7% and 37.1%) is achieved. In addition, more interconnections (14.3% and 15.5%), which violate the delay constraints, are corrected by our buffer planning method.

The rest of this paper is organized as follows. Section 2 formulates the fixed-outline floorplanning as well as buffer insertion. In Section 3, Sliced-LFF method is introduced. Section 4 shows how to integrate buffer planning into LFF. Section 5 discusses the net-classing strategy and our post greedy method while Section 6 shows the experiment results. At last, conclusions and future work are given in Section 7.

2. Problem Formulation

The input of the algorithm is a set of N rectangular IP modules $M = \{m_1, m_2, \ldots, m_N\}$ and their areas, locations of I/O pins, netlist, and delay constraints of the nets. Moreover, the width W and the height H of the chip are also given as the fixed-outline constraint.

The output is the coordinates of the bottom-left corners of the modules and their orientations. Meanwhile, deadspace is reserved for buffers on the resultant floorplan. Assumptions in this paper are listed as follows.

(i) All the modules are hard modules. They have fixed sizes and aspect ratios.

(ii) All the buffers can only be inserted in the deadspace among the modules instead of the inside of modules.

(iii) All nets are 2-pin, and multipin nets are split to 2-pin nets, which is the same as previous works [3, 12].

(iv) All the modules can be rotated or mirrored during the packing process.

The goal of our algorithm is to find a feasible packing in the fixed outline, which is able to reserve deadspace for the required buffers as many as possible.

3. Sliced LFF Algorithim

3.1. Original LFF Algorithm. LFF is a published floorplanning algorithm [15], which works with the fixed-outline constraint. A typical LFF packing process is demonstrated in Figure 1.

In Figure 1, modules are placed from four corners of the fixed-outline gradually to the center one by one. To place one module (or called a *packing step*), the most suitable candidate from the queue of unplaced modules will be selected and placed on the most suitable corner. "Suitable" is quantified by definitions of "flexibilities". Two essential flexibilities are defined by modules' sizes (R_{if}) and their interconnections (FC_i) to ensure that modules with larger sizes or with more interconnections to the placed modules will have higher

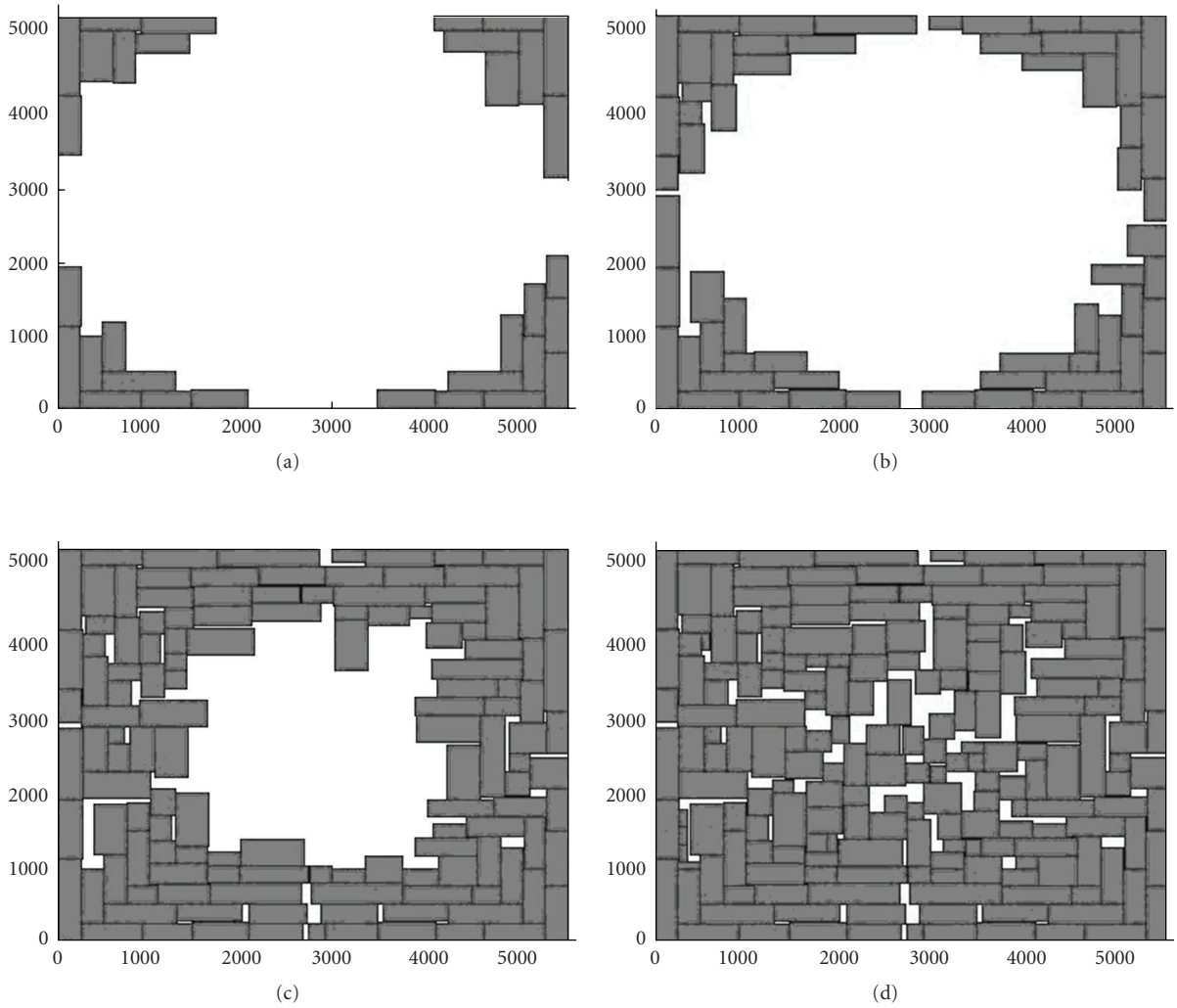

Figure 1: A typical LFF packing process.

priorities to be placed. More specifically, the suitability (or called *Fitness Value*, FV) of unplaced Module i is defined by the weighed sum of R_{if} and FC_i as in (1):

$$\mathrm{FV}_i = w_1 \cdot R_{if} + w_2 \cdot \mathrm{FC}_i, \tag{1}$$

where $R_{if} = r_1 \times (1 - a_i/(W \times H)) + r_2 \times (1 - \max(w_i, h_i)/\min(W, H))$,

$$\mathrm{FC}_i = 1 - \sum_{j=0, j \neq i}^{N} \left(\frac{W_{ij}}{W_{\mathrm{net}}} \right). \tag{2}$$

In (1), a_i, w_i and h_i are the area, width, and height of Module i. W and H are the width and height of the fixed outline. r_1, r_2, w_1, and w_2 are weight factors. W_{ij} denotes the number of nets between Module i and placed Module j while W_{net} means the total number of nets.

3.2. Sliced-LFF Algorithm. Original LFF algorithm has to store two lists. One is for all the unplaced modules, and the other is for all the corners on the current floorplan. In each packing step, the most suitable module on the most suitable corner will be chosen and placed. Therefore, LFF evaluates coordinates of the corners, modules' sizes, and their interconnections. But it has no consideration on the shape of deadspace. Unfortunately, in some specific cases, ignoring this information will cause a packing error. Figure 2(a) gives an example.

In Figure 2(a), Module e is going to be placed and there are 12 corners in the current floorplan, which are recorded in the LFF process. However, we find that Module e cannot be placed on any of these 12 corners. In this case, it will finally fail to pack Module e.

Nevertheless, we know that Module e can be placed like Figure 2(b). A further observation shows that LFF packing will become more robust if it does not store all the corners but catches the shape of deadspace. In this paper, an effective model called HSlice and VSlice structures is introduced to partition the deadspace. As shown in Figure 3, the whole deadspace is cut into five slices horizontally and

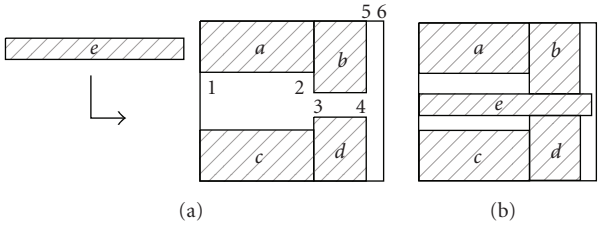

Figure 2: A Packing error in original LFF.

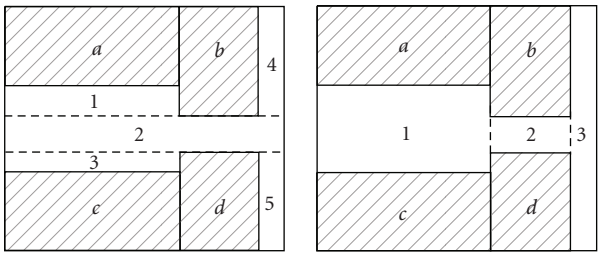

Figure 3: HSlice and VSlice structures.

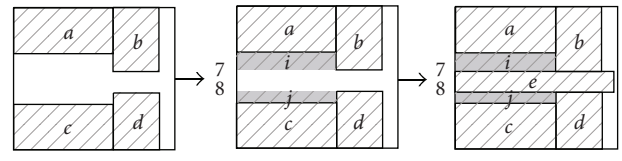

Figure 4: A Slice-based packing with fictitious modules.

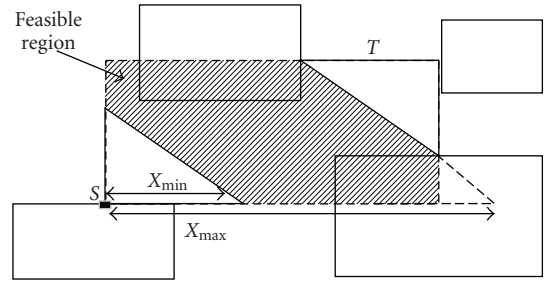

Figure 5: 2-D feasible region.

three vertically. With their help, we can take more advantages of deadspace and finally correct the error in Figure 2(a), as shown in Figure 4. In our implementation, the list of HSlice and VSlice is stored instead of the list of all the corners.

Another observation is that all the corners can be divided into two categories, depending on whether the two edges of this corner belong to different modules or not. If they do, the corner is named 90-degree corners, for example, Corners 1, 2, 5 and 6 in Figure 2(a); and if not, it is called 270-degree corners, for example, Corners 3 and 4. Furthermore, we find that higher area utilization will be achieved if modules are only arranged on 90-degree corners.

After HSlice and VSlice structures are obtained, coordinates of all 90-degree corners will be calculated by the following criterion using HSlice and VSlice.

A vertex will be a 90-degree corner if and only if

(i) this vertex belongs to one HSlice and one VSlice at the same time;

(ii) the direction of this vertex (i.e., top-left, top-right, bottom-left and bottom-right) should be the same in both HSlice and VSlice.

After the coordinates of all the corners are calculated, Module *e* will be successfully placed on a newly generated Corner 7 after defining HSlice 1 and 3 as two fictitious Module *i* and *j*, as shown in Figure 4. Furthermore, HSlice and VSlice structures will provide more information of the deadspace and facilitate deadspace planning for buffers.

4. Buffer Insertion

In order to integrate buffer insertion issue into LFF packing, we first adopt two published models on buffer insertion: FR and IFR. Based on these two models, some detailed techniques are developed for this integration. Then, how to implement the buffer insertion process in LFF will be summarized in Section 4.5. In Section 4.6, we will illustrate how the geometric structure impacts the success rate of buffer insertion.

4.1. The Notion of FR and IFR. Buffer insertion spots have their own constraints to meet the timing requirement. Basic studies on these constraints are listed as follows.

In this paper, we adopt 2D Feasible Region (FR) and Independent Feasible Region (IFR) to calculate buffer insertion spots. 2-D FR is a polygon consisting of all the feasible spots for a buffer of a net. IFR of a buffer means a region where this buffer can be placed once the other buffers of the same net have been inserted successfully in their IFRs. Figure 5 shows the 2-D Feasible Region of a 2-pin net.

For each net, inserting more buffers may not always improve the interconnect delay. There is a tradeoff between wire delay and buffer delay. That means that there is an optimal number for the buffers, which will minimize the interconnect delay of this net. We assume that this number is N and the delay with optimal number of buffers is T^N_{opt}. In our implementation, we let N vary from zero to a threshold (e.g., 30) for each net. Then, we calculate the interconnect delay with each N and record the smallest one as T^N_{opt}.

Equations to calculate the delay with N buffers can be referred in [16]. We omit them for the conciseness of this paper, since they are pretty complicated and hard to explain.

After that, all the nets can be divided into three groups and their definitions are discussed as follows.

Definition 1. Suppose that the Elmore delay and target delay of a 2-pin net are T^N_{elmore} and T^N_{tgt}, respectively. The optimal

delay with best buffer insertion is T^N_{opt}. After that, all the nets could be divided into 3 classes.

(i) Successful Nets (SNs). If either $T^N_{\text{elmore}} \le T^N_{\text{tgt}}$ or $T^N_{\text{elmore}} > T^N_{\text{tgt}} \ge T^N_{\text{opt}}$ is satisfied by successful buffer insertion, the target delay for the net will be achieved and the net is called Successful Net.

(ii) Failed Nets (FNs). If $T^N_{\text{elmore}} > T^N_{\text{tgt}} \ge T^N_{\text{opt}}$ and the buffer insertion is failed (the entire FR has been taken up by IP modules), the target delay for the net will not be achieved and the net is named Failed Nets.

(iii) Invalid Nets (INs). If $T^N_{\text{tgt}} < T^N_{\text{opt}}$, this net will not meet the target delay even by buffer insertion and the net is defined as Invalid Nets.

The goal of buffer insertion is to increase the number of SN but to decrease the number of IN and FN.

4.2. Flexibilities' Definition for Buffer Insertion. Since LFF packing process runs with the guidance of the flexibility, if we intend to consider buffer insertion issue in LFF, definitions of flexibilities for buffer insertion will be necessary. Since IN and FN should be treated differently and accordingly, we should define two flexibilities in LFF packing.

Definition 2. As listed in Section 4.1, when packing Module i, three classes of nets are generated: SN, FN, and IN. Moreover, their amounts are denoted by n^i_{SN}, n^i_{FN}, and n^i_{IN}. Meanwhile, the number of nets that are related to Module i is denoted by n^i_{NET}.

(1) Flexibility for FN of Module i (F^i_{FN}) is as follows:

$$F^i_{\text{FN}} = \frac{n^i_{\text{FN}}}{n^i_{\text{NET}}}, \tag{3}$$

where n^i_{FN} of Module i has two origins: one is the nets of Module i whose IFRs are blocked by other placed modules and the other is the nets of placed modules whose IFRs are blocked by Module i.

(2) Flexibility for IN of Module i (F^i_{IN}) is as follows:

$$F^i_{\text{IN}} = \frac{n^i_{\text{IN}}}{n^i_{\text{NET}}}, \tag{4}$$

By far, the cost function for Module i used in LFF can be extended from (1) to (5):

$$\text{FV}_i = w_1 \cdot R_{if} + w_2 \cdot \text{FC}_i + w_3 \cdot F^i_{\text{FN}} + w_4 \cdot F^i_{\text{IN}}, \tag{5}$$

4.3. True-Deadspace Based on the Guidance of FV. Figure 6 shows two packing candidates. Obviously, FV_c of placing Module c at Corner 1 will be less than Corner 2. because placing Module c at Corners 2 will block three buffers. As a result, Corner 1 has less flexibility. According to the guidance of FV_c, Module c will be placed at Corner 1 instead of Corner 2. Thus, the three buffers will be protected in the deadspace between Module a and c. This kind of deadspace is called *True-Deadspace*, corresponding to *Pseudo-Deadspace* to be defined in Section 4.4.

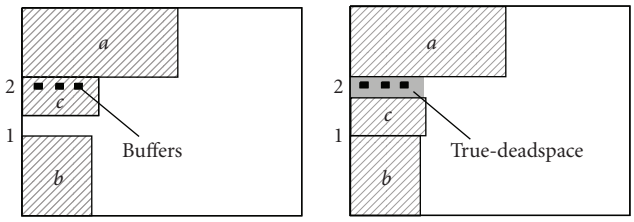

Figure 6: True-deadspace based on the guidance of FV.

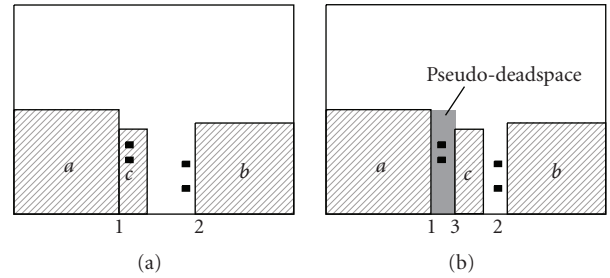

Figure 7: The Function of pseudo-deadspace.

4.4. Generating Pseudo-Deadspace. Figure 7 illustrates one fact that only using FV to guide the placement of modules in LFF Algorithm is not enough.

Notice that a corner should be always needed before placing a module in LFF. Thus, in Figure 7(a), Module *cc* should be placed at neither Corner 1 nor 2. However, both Corner 1 and 2 are worse than the placement in Figure 7(b).

LFF method only allows modules to be placed at one existing corner. Therefore, in order to generate such a placement as Figure 7(b), Pseudo-Deadspace is introduced and placed at Corner 1 as a fictitious module. With its help, Corner 3 is newly created and Module c could be placed at Corner 3. After that, the better placement in Figure 7(b) is finally achieved.

4.5. Buffer Insertion Implementation. The methods in Sections 4.3 and 4.4 have done a good allocation of deadspace for buffer insertion. Then in LFF packing process, buffers are inserted simultaneously with packing the modules. If one module is placed, its buffers related to this module are also inserted in one spot of their IFRs. However, how to select a suitable buffer spot in its IFR also needs discussions.

In the algorithm, *Pseudo-Deadspace* is treated as a fictitious module. Therefore, if buffers are inserted far from modules, the *Pseudo-Deadspace* will be huge, which will reduce the area utilization of the chip. As shown in Figure 8, Buffers 1 and 2 are far from Module a and Huge *Pseudo-Deadspace* is generated

Since buffers should be inserted close to modules to reduce the size of Pseudo-Deadspace, we use *Laplacian Operator* that is widely used in Digital Image Processing for edge detection of placed modules and finally insert buffers along the edges of modules.

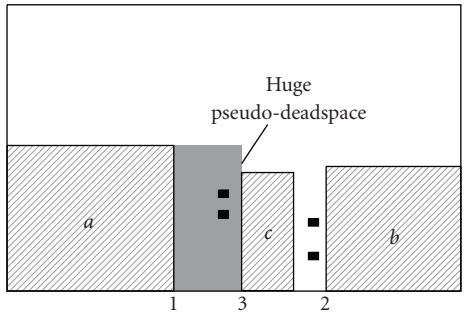

Figure 8: Huge pseudo-deadspace.

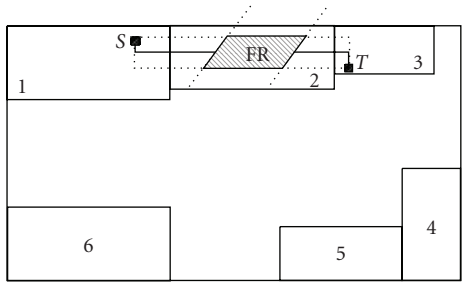

Figure 9: The noncausality during module placement.

4.6. A Comparison between SA and LFF. SA is usually based on topological representations of a floorplan. However, besides different topologies, we notice that buffer insertion is also sensitive to different geometric structures with the same topology. For example, in Figure 7(a), the placement will block two buffers while the one in Figure 7(b) blocks none. Considering that both of them have the same topological structure, it means that topological representation in SA cannot distinguish the difference between Figures 7(a) and 7(b). Some critical geometric information for buffer insertion will be lost.

On the contrary, LFF-based algorithm can tell the geometric difference within the same topology, like Figures 7(a) and 7(b). The packing order of LFF can also be optimized to decide a suitable topological structure among the modules. Thus, both geometric and topological information will be considered for buffer insertion in LFF, which is the motivation of our work.

5. Improved Buffer Insertion

LFF has no backtracks in its packing process. Therefore, further studies of LFF packing process will come up with a problem called *Noncausality*. To alleviate the impact of this problem, a net-classing strategy is put forward according to different characteristics of the nets. Moreover, a 2-staged LFF method and a post greedy algorithm are developed based on this net-classing strategy.

5.1. The Challenge in LFF Packing Process. The process of the LFF algorithm is to place the modules one by one. When considering buffer insertion issue, one problem will come up, as illustrated in Figure 9.

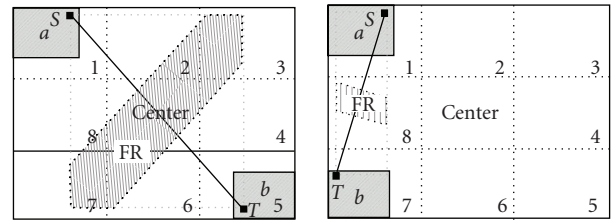

Figure 10: Two classes of nets.

Table 1: Characteristics of two classes of nets.

	Amount	Buffers needed per net	FR's vitality
CCN	Fewer	More	Stronger
CNN	More	Fewer	Weaker

In Figure 9, Modules 1, 2, and 3 are placed in order. The FR between Module 1 and 3 is occupied by Module 2. However, it is impossible to consider this case when placing Module 2, because Module 3 has not been placed at that time and this FR between Module 1 and Module 3 cannot be calculated in advance. This is called *Noncausality*.

If we can predict Module 3's location, the effect of *Noncausality* can be greatly handled. However, because of the complexity of the FRs, such a precise prediction is usually hard to implement. It means that completely avoiding the impact of *Noncausality* is not practical. It is a *chicken-egg* problem. However, we propose an effective method to alleviate this impact.

5.2. Net Classing Strategy. Fortunately, we find that the impact of *Noncausality* differs from two different classes of 2-pin nets. In one class, the line from source to sink has a slope close to +1 or −1 and these 2 pins are far from each other. Therefore, they have larger FRs than the other group of nets, as mentioned in Figure 10(a). Larger FRs mean stronger vitalities and it is easier to find a new spot candidate in such a large FR even if a placed module may occupy the original spot. Since the line from their source to sink usually crosses the center of the chip, they are called *Center-Crossing Nets* (CCNs) while the other class is named *Center-Noncrossing Nets* (CNNs). Table 1 shows their characteristics.

In Table 1, the number of CCN is usually smaller than CNN due to wirelength optimization. Meanwhile, CCN has longer distance from source to sink and will need more buffers than CNN per net. In this paper, a quantitative definition of CCN and CNN will be needed.

Definition 3 (*Center-Crossing* and *Center-Noncrossing Nets*). In Figure 10, the whole chip is cut into nine grids. In our implementation, nine is enough to define the difference between CCN and CNN. The middle one is marked as Center while the others are numbered with 1 to 8 clockwise around the Center.

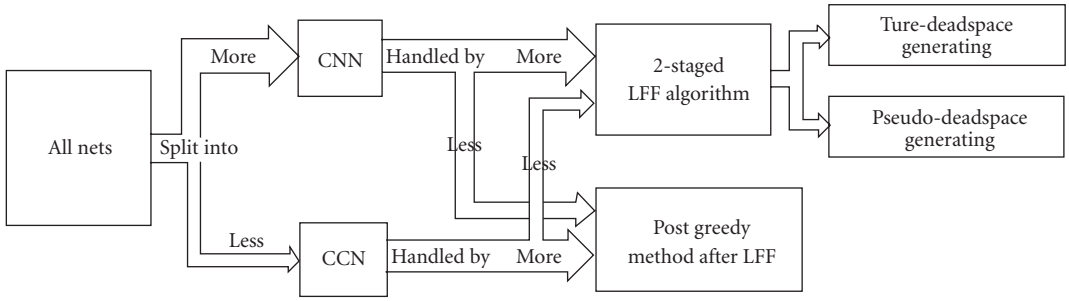

Figure 11: The Relation of All the Techniques in This Paper.

For a 2-pin net, suppose that its source and sink pins locate in *Region* N_S and *Region* N_T, respectively, if any of the three conditions

$$(N_S + 3) \bmod 8 = N_T,$$
$$(N_S + 5) \bmod 8 = N_T, \tag{6}$$
$$((N_S + 4) \bmod 8 = N_T) \wedge (N_S \bmod 2 = 1),$$

is met, the net will be defined as CCN, for example, the one in Figure 10(a). Otherwise, it is called as a CNN, for example, the one in Figure 10(b).

All nets of a circuit will be divided into these two classes at last. Based on the different impacts of *Noncausality* between CCN and CNN, a 2-staged LFF method and a postmethod are put forward.

5.3. Algorithm Framework: 2-Staged LFF. Figure 10 shows that the FRs of a CCN are much larger. So they have more buffer insertion candidates than CNN.

Meanwhile, since the modules are placed one after another, the chief objective in (5) could be different as the packing step continues. For example, at the beginning of packing, we focus mainly on the area utilization, wirelength and so on. When only a few modules are left unplaced, how to place them in the fixed-outline constraint and maximize the convergence of our algorithm becomes the key issue.

Similarly, in buffer planning process, the algorithm is segmented into two stages. In Stage 1, a more strict constraint is adopted to improve the success rate of buffer insertion mainly for CNNs, whose FRs are smaller and harder to insert buffers. In Stage 2, since fewer modules are left unplaced and the chief goal is shifted to improve the success rate of fixed-outline floorplanning, a more tolerant rule will be adopted for buffer insertion. Therefore, the success rate of buffer insertion will shrink at that time. However, buffer failures in this period mostly come from CCNs, which have larger FRs and are easier to find an alternative insertion spot. When the LFF ends, the post floorplanning procedure will be invoked to save these failed CCNs.

5.4. A Post Greedy Method after LFF Packing. Since 2-staged LFF in Stage 1 ensures a high success rate of CNNs and leaves many failed CCNs in Stage 2, a postmethod is developed. The goal of this post floorplanning method is to increase CCNs' success rate on buffer insertion since their buffers have stronger vitalities.

In this method, a queue of all failed buffers is set up. Buffers with smaller FR and fewer insertion candidates will be inserted in the foreside of the queue. Then we pick up one buffer from the queue and search a new spot in its IFR to insert it, and the next buffer follows. Using this greedy method, we save failed buffers as many as possible.

The experiments in Section 6.2 show that lots of failed nets will be saved by this greedy method.

5.5. The Relation of All Algorithm Details. Finally, all the techniques discussed in Sections 3, 4, and 5 are integrated in Figure 11. "More" and "Less" describe the importance of these techniques to the two classes of nets. For instance, more failed CNNs are saved by our 2-staged LFF procedure than the Post Greedy Method. In other words, the 2-staged LFF is more important than the greedy method for CNNs. Thus, the arrowhead from CNN to 2-staged LFF is thicker than the one from CNN to our Post Greedy Method.

Besides, a flow chart of our algorithm is shown in Figure 12. Our algorithm consists of two parts: 2-staged LFF and a Post Greedy Method. The threshold for the two stages of LFF is 20%. That means that Stage 2 will be started if 80% of circuit modules are successfully packed. The algorithm will return false if no feasible solutions could be found in each packing step.

6. Experimental Results

Our work is implemented using ANSI C and all the experiments are completed on a 1.6-GHz Intel PC with 1 GB memory. The weight factors w_1, w_2, w_3, and w_4 in (5) are set to 0.3, 0.3, 0.2, and 0.2, respectively.

6.1. Comparison Experiments. In this paper, we integrate buffer insertion issue into fixed-outline floorplanning. To test the result of our buffer planning algorithm, an SA-based outline-free floorplanning algorithm with buffer planning [12] is adopted as the comparison work. Experiments are finished to explain the comparison between our LFF packing algorithm (marked as F2) and the SA-based work (marked as F1) [12].

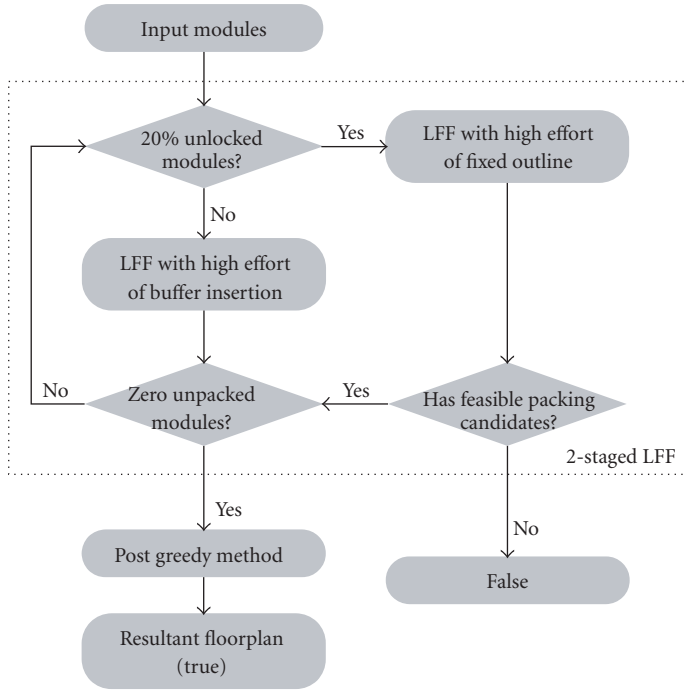

Figure 12: Flow chart of the algorithm.

We use six MCNC standard benchmarks and one generated benchmark M198. All the multipin nets are split into 2-pin nets just like [12]. All the power and ground interconnects are ignored. To generate the delay constraints, a floorplan is acquired by running a common SA-based floorplanner at first. Then for each net, we compute its T^N_{opt} and assign the target delay as $1.1 * T^N_{\text{opt}}$. All the pretreatments in this section are kept the same as the comparison floorplanner F1 [12].

The results are listed in Table 2(a) and 2(b) with different needs of buffers in different feature sizes. Table 2(b) does not include the first three benchmarks in Table 2(a), because they are not included in [12] either.

Our work achieves 40.7% and 37.1% improvement on buffer insertion rate, compared with the SA-based work [12]. As a result, the number of FN has been decreased while the number of SN has been increased. Meanwhile, our algorithm is more than 100 times faster than [12], since it is deterministic without any iterations or packing backtracks.

In Table 2(a) and 2(b), the fixed-outline constraint with larger area than F1 is required, because more buffers are inserted in our LFF method and they should consume more deadspace. However, F2 has similar wirelength as F1.

From Table 2(a), we can also find that the increase on the number of SN achieved by our work will be abated when the benchmark has fewer modules. For example, in Xerox of Table 2(a), there is even a decrease on SN compared with F1 [12]. This is because the solution space is more discrete when the floorplanner has fewer modules to operate. That means that too few modules may not provide enough floorplanning candidates to make better use of the deadspace. However, our floorplanner is usually invoked in hierarchical design. So, the number of modules in each hierarchy could totally be decided using a different threshold setting in the partitioning stage. We can choose a number, which is not quite small and easier for our buffer planning algorithm.

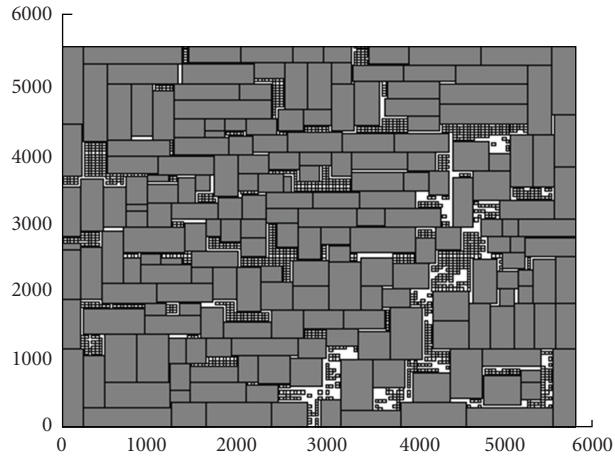

Figure 13: Floorplan of M198 with buffers.

At last, a practical floorplan for M198 is presented in Figure 13.

6.2. Experiments on Improved Buffer Insertion. In this section, detailed experiments on M198 are performed to explore the internal mechanism of LFF packing. It shows that the efficiency of basic methods in Section 3 and 2-staged LFF based on net-classing strategy in Section 4, as in Table 3.

Table 2

(a) The comparison between the SA-based and LFF-based floorplanner

	Area (mm^2)		Wirelength (mm)		#Inserted Buffer/#Buffer		#Successful Nets		#Failed Nets		Timing Slack per FN (ps)		Time(s)	
Methods	F1	F2	F1	F2	F1	F2	F1	F2	F1	F2	F1	F2	F1	F2
Apte	48.32	53.25	459.6	563.7	37/123	102/116	103	118	52	18	23.4	20.2	46	0.15
Xerox	86.20	92.00	671.5	783.8	114/203	247/265	388	349	39	15	32.1	28.9	109	0.3
Hp	39.56	40.56	216.1	196.0	60/121	60/87	145	160	58	17	19.1	18.8	35	0.2
Ami33	32.00	33.50	96.7	110.9	194/386	164/176	204	247	20	8	18.7	16.7	26	1.3
Ami49	172.2	158.8	1489.0	1462.8	277/545	435/447	330	448	161	30	43.0	41.2	760	3.7
Playout	460.6	470.0	12053	12970	613/1316	806/924	1564	1788	365	71	56.4	52.5	2330	46.6
M198	34.52	33.08	618.0	491.1	594/1232	741/833	1587	1813	536	157	17.8	14.9	2538	120.6
Average	+2.03%		+3.74%		+40.7%		+14.3%		−70.3%		−10.3%		×163	

(b) The Comparison between the SA-based and LFF-based Floorplanner with larger needs of buffers

	Area (mm^2)		Wirelength (mm)		#Inserted Buffer/#Buffer		#Successful Nets		#Failed Nets		Timing Slack per FN (ps)		Time(s)	
Methods	F1	F2	F1	F2	F1	F2	F1	F2	F1	F2	F1	F2	F1	F2
Ami33	33.50	34.00	99.1	104.3	236/409	339/354	196	227	32	11	14.3	13.8	396	1.62
Ami49	162.0	162.4	1507.0	1505.2	364/892	812/958	299	330	184	49	29.2	25.7	890	4.74
Playout	460.6	467.7	12363	12189	1240/2825	1597/1972	1230	1421	395	265	35.4	32.8	2920	52.5
M198	33.24	34.20	606.2	523.3	1329/3559	1677/2513	1240	1493	445	351	15.6	14.3	3975	138.3
Average	+1.54%		−1.42%		+37.1%		+15.5%		−48.3%		−8.6%		×129	

Table 3: Detailed results of M198 floorplanning.

	CCN		CNN	
	2-Staged LFF	Postmethod	2-Staged LFF	Postmethod
% of #Nets	7.4%		92.6%	
% of #Buffers	22.2%		77.8%	
#SN	62	134	1505	1704
#FN	76	4	220	21
#SN/(#FN+#SN)	44.9%	97.1% (+53.8%)	87.2%	98.8% (+11.6%)

(1) Characteristics of the 2 Classes of Nets. In Table 3, the percentages of CCNs and CNNs are 7.4% and 92.6%, since fewer CCNs are helpful to reduce the chip wirelength. However, CCN usually needs more buffers than CNN. The percentages of CCNs' and CNNs' buffer needs are 22.2% and 77.8%, which proves the description of the two classes of nets in Table 1.

(2) Collaboration of Methods in Figure 11. Moreover, Table 3 shows that the strategy listed in Section 4 is useful. In 2-staged LFF, CNNs are better settled and the ratio between SN and FN is 1505 : 220. The ratio is much higher than CCN (i.e., 62 : 76).

When LFF packing ends, the post greedy method is invoked. The ratio 62 : 76 has been increased to 134 : 4. Compared to CNN's increase (+11.6%), the post greedy method in Stage 2 is more important for CCN (+53.8%), which also proves the conclusion in Figure 11.

7. Conclusions and Future Work

In this paper, we develop a more robust version of LFF called Sliced-LFF and integrate buffer planning into LFF packing with a fixed-outline constraint. By using net-classing and 2-staged LFF strategy, the impact caused by *Noncausality* during module placement is greatly alleviated. A great improvement on the success rate of buffer insertion (40.7% and 37.1% in different feature sizes) is achieved, compared with an SA-based previous work.

Currently, all the nets are handled with the same priority. In practical designs, some nets are more important than the others, such as clock tree. The priority of the nets will be considered in future work.

References

[1] J. A. Roy, S. N. Adya, D. A. Papa, and I. L. Markov, "Min-cut floor-placement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 7, pp. 1313–1326, 2006.

[2] S. N. Adya and I. L. Markov, "Fixed-outline floorplanning: enabling hierarchical design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 6, pp. 1120–1135, 2003.

[3] J. Cong, T. Kong, and D. Z. Pan, "Buffer block planning for interconnect-driven floorplanning," in *Proceedings of*

the IEEE/ACM International Conference on Computer-Aided Design (ICCAD '99), pp. 358–363, November 1999.

[4] P. Sarkar, V. Sundararaman, and C. K. Koh, "Routability-driven repeater block planning for interconnect-centric floorplanning," in *Proceedings of the International Symposium on Physical Design (ISPD '00)*, pp. 186–191, April 2000.

[5] X. Tang and D. F. Wong, "Planning buffer locations by network flows," in *Proceedings of the International Symposium on Physical Design (ISPD '00)*, pp. 180–185, April 2000.

[6] F. F. Dragan, A. B. Kahng, I. Mandoiu, S. Muddu, and A. Zelikovsky, "Provably good global buffering using an available buffer block plan," in *Proceedings of the IEEE/ACM International Conference on Computer Aided Design (ICCAD '00)*, pp. 104–109.

[7] C. W. Sham and E. F. Y. Young, "Routability driven floorplanner with buffer block planning," in *Proceedings of the International Symposium on Physical Design (ISPD '02)*, pp. 50–55, April 2002.

[8] F. Rafiq, M. Chrzanowska-Jeske, H. H. Yang, and N. Sherwani, "Inegrated floorplanning with buffer/channel insertion for bus-based microprocessor designs," in *Proceedings of the International Symposium on Physical Design (ISPD '02)*, pp. 56–61, April 2002.

[9] C. J. Alpert, J. Hu, S. S. Sapatnekar, and P. G. Villarrubia, "A practical methodology for early buffer and wire resource allocation," in *Proceedings of the 38th Design Automation Conference (DAC '01)*, pp. 189–194, June 2001.

[10] S. Chen, X. Hong, S. Dong, and Y. Ma, "A buffer planning algorithm based on dead space redistribution," in *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC '03)*, pp. 435–438, Kitakyushu, Japan, January 2003.

[11] I. H.-R. Jiang, Y.-W. Chang, J.-Y. Jou, and K.-Y. Chao, "Simultaneous floorplanning and buffer block planning," in *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC '03)*, pp. 431–434, Kitakyushu, Japan, January 2003.

[12] Y. Ma, X. Hong, S. Donag, S. Chen, C. K. Cheng, and J. Gu, "Buffer planning as ah integral part of floorplanning with consideration of routing congestion," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 4, pp. 609–620, 2005.

[13] Q. Dong, B. Yang, J. Li, and S. Nakatake, "Incremental buffer insertion and module resizing algorithm using geometric programming," in *Proceedings of the 19th ACM Great Lakes Symposium on VLSI (GLSVLSI '09)*, pp. 413–416, May 2009.

[14] X. He, S. Dong, Y. Ma, and X. Hong, "Simultaneous buffer and interlayer via planning for 3D floorplanning," in *Proceedings of the 10th International Symposium on Quality Electronic Design (ISQED '09)*, pp. 740–745, March 2009.

[15] S. Dong, X. Hong, Y. Wu, Y. Lin, and J. Gu, "VLSI block placement using less flexibility first principles," in *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC '01)*, pp. 601–604, Yokohama, Japan, January 2001.

[16] C. Alpert and A. Devgan, "Wire segmenting for improved buffer insertion," in *Proceedings of the 34th Design Automation Conference*, pp. 588–593, June 1997.