國 立 清 華 大 學

博 士 論 文

先進微影技術與三維整合技術之優化方法

Optimization Methods for Modern Lithography and 3D Integration Technologies

系 所 別 ： 資訊工程學系 組別：
學號姓名 ： 938316 蔡名詔 (Ming-Chao Tsai)
指導教授 ： 黃婷婷 博士 (Dr. TingTing Hwang)

中華民國九十九年十二月

Optimization Methods for Modern Lithography and 3D Integration Technologies

Student: Ming-Chao Tsai

Advisor: Prof. TingTing Hwang

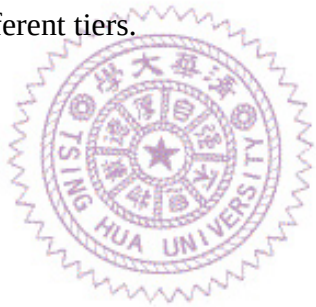
December 3, 2010

Abstract

As the semiconductor technology advances, the interconnect delay gradually dominates the entire circuit delay and becomes the bottleneck of the circuit performance. To further improve the performance, manufacturers invest great deal of effort to reduce the delay by miniaturizing the chip size and shortening the interconnects. In this dissertation, we propose using modern lithography and 3D integration technologies to scaling down the chip size. Since the phase shift mask(PSM) is a very effective lithography technology to miniature the layout patterns, we focus on PSM design issues and propose a wire spreading algorithms to modify layouts for PSM compliance. Experimental result shows that our algorithm can eliminate more than 98% of phase conflicts without increasing the die size.

With the aid of through-silicon via (TSV), 3D integration is able to shorten the wirelength of inter-tier net and achieves high performance. However, TSV is not volumeless point and cannot be placed anywhere on a layout. Without planning TSVs in the early design stage, a post TSV insertion procedure is required to arrange TSV to white space. To this end, we also propose a 3D floorplanning algorithm to simultaneously plan functional blocks and TSVs. Experimental results show that our algorithm outperforms a post-processing TSV planning algorithm in wirelength by 22.3%.

Although TSV potentially reduces the wirelength of a 3D-IC, the area overhead of TSV poses negative impact to circuit. Applying too many TSVs in a design increases the size of a 3D-IC and extends the distances among active devices. Therefore, we also propose evaluation methods to study the trade-off among wirelength, number of TSVs, size of TSVs and placement of 3D-ICs. Experimental results reveal that the optimal number of TSVs of a design varies with the size of TSVs. When the size of TSV is small, the using more TSVs is beneficial for wirelength reduction. On the contrary, when large TSV is applied, a design prefers using routing topologies with least number of TSVs to minimize the total wirelength. Also, our experimental results show that the best partition scheme for placement is sensitive to the size of TSV. The larger TSV we use, the earlier we have to partition cells to different tiers.



Contents

List of Figures

List of Tables

Chapter 1

Introduction

Since the invention of semiconductor transistors, the performance of integrated circuits (ICs) keeps improving with the evolving of semiconductor technology. The ever-growing demand for higher performance drives the development of semiconductor industry. To reduce the interconnect delay and achieve higher circuit performance, IC manufacturers invest a great deal of effort into IC miniaturization technologies. Among all the technologies, the advanced lithography and three dimensional (3-D) integration technologies are viewed as two promising solutions to effectively scale down circuits. Phase shift mask (PSM) is the pioneer of the advanced lithography technology and has been widely used in recent years. By effectively scaling down the feature size and shortening interconnects, PSM drastically improves the performance of a chip. Different from PSM, the 3-D integration technology stacks up multiple chips and uses through-silicon vias (TSVs) to connect the inter-chip wires. The TSVs directly punch through silicon substrates and provide direct links among the stacked up chips. Hence, the signal delay can be reduced. Since these two technologies are in different stages of the manufac-

turing flow and can be used without interfering each other, applying both of the technologies further improves the performance.

Although PSM and 3D integration technologies profoundly enhance the performance of circuits, they also bring extra burdens to modern circuit design. In order to apply PSM, the layout of a circuit must comply with extra design constraints. Thus, the design rules become more complicated. Also, in the 3D integration, TSVs occupy the silicon area and are viewed as routing obstacles. Applying too many TSVs in a design, the negative impact will cancel out the benefit of 3D integration. To take full advantage of the modern techniques, it is necessary to make an in-depth study on PSM and 3D integration technologies and designs optimization methods for them.

This dissertation is divided into three parts. The first part focuses on the PSM technology. We propose algorithms to modify the layout for PSM rule compliance. The second part studies the 3D floorplan with TSV planning consideration. Since lengths of interconnects change with the areas and positions of TSVs, we propose an algorithm to simultaneously plan TSVs and functional blocks for wirelength reduction. As mentioned previously, TSVs potentially reduce the wirelength by providing direct links in 3D-IC. However, the area overhead of TSVs poses negative impact to designs. To make a comprehensive analysis, in the third part of the dissertation, we study the trade-off among wirelength, number of TSVs and size of TSVs. Also, we propose a fast evaluation algorithm to evaluate how TSVs influence the wirelengths of nets with different routing topologies.

The rest of the dissertation is organized as follows. Chapter 2 presents a wire

spreading method for PSM-aware layout modification. Chapter 3 presents TSV planning in 3D-ICs. Chapter 4 presents the evaluation of trade-off among wire-length, number of TSV and placement in 3D-ICs. Finally, Chapter 5 concludes our work.

Chapter 2

An MILP-Based Wire Spreading Algorithm for PSM-Aware Layout Modification

Optical lithography has been a critical step in the VLSI fabrication process. As the pitches of leading-edge products scale down, phase shifting mask (PSM) and immersion lithography are viewed as potential solutions to carry the 193nm lithography beyond 65nm node. The key of immersion lithography lies on high-index fluids which increase the numerical apertures of the lithography system. Nevertheless, ASML and Nikon recently announced that the numerical apertures had essentially reached their limit for water-based immersion lithography [1]. In the 45nm technology node, the pitch of metal 1 (M1) is reduced to 90nm, which necessitates the incorporation of immersion lithography and strong PSM techniques such as alternating PSM (altPSM). Assume that the minimum spacing which can be resolved by applying conventional mask is B. Beyond this spacing, strong constructive diffraction effect interferes the imaging of critical features. Applying

altPSM extends the limit of resolution to b $(= B/2)$ by destructive interference. However, this resolution improvement can be achieved only if the apertures of two adjacent critical features are assigned to opposite phases. Although altPSM shows great potential in resolution enhancement, a layout must be compliant to the phase assignment constraint. Figure 2.1 shows a layout which cannot satisfy the phase assignment constraint. Since spacing between any two of the three rectangular wire segments in Figure 2.1 is critical, any two of these features must be in opposite phases. However, no matter what phase we assign to the bottom feature, it will be the same as one of the upper two features. Thus, this layout has the phase conflict problem. Masks can be categorized into dark field masks and bright field masks. The dark field masks are mainly used for metal layers, and the bright field masks are usually used for poly layers. Targeting on the dark field altPSM, McCullen reported routing restrictions that enable the generation of phase-correct layout [2]. Berman *et al.* [3] proposed a graph based algorithm for solving the phase conflict problem. To obtain altPSM compliant layouts, conflict graphs are constructed according to given layouts, and then a minimum-weight set of edges is removed from each graph to ensure the resultant graph 2-colorable. The edge deletions in the conflict graphs are accomplished by changing the placement of the features. Although their algorithm is efficient, it is hard to assign edge weights since we cannot compute how far each layout object needs to be moved before replacement is done. Moreover, this approach may induce area overhead. To handle the phase conflict problem for bright field masks, Cao et al. proposed a Boolean satisfiability based method to generate PSM compliant and composiable

cell libraries [4]. Although their algorithm considered the phase conflict problem within and between library cells, it does not consider phase conflicts among interconnections. In addition, the areas of resultant library cells increase. Chiang et al. proposed a layout correction algorithm for standard-cell layouts [5]. The proposed algorithm targets on bright field AAPSM (alternating aperture PSM) for the poly layer. It inserts end-to-end spaces into layouts to solve the phase conflict problem. Although it completely eliminates phase conflicts in a given layout, unfortunately it also induces area increasing. Since the pitch of critical metal layers continues to scale down, it becomes urgent to find a solution to solve the phase conflict problem for critical metal layers. In addition, the advanced VLSI technology adopts the dual damascene (DD) process flow to accomplish the copper metallization. Therefore, in this chapter, we focus on the dark field altPSM which pertains to the DD process. Although Berman *et al.* [3] has presented an efficient algorithm to remove phase conflicts, unfortunately the algorithm not only induces area overhead but also changes the placement, which is costly from the viewpoint of a physical design flow. To cope with these drawbacks, we propose a new layout modification algorithm to solve the phase conflict problem. Our algorithm uses the wire spreading technique to adjust the wire segment positions on the critical metal layers within the predefined die size. In addition, to effectively reduce the perturbation to a layout, our algorithm tries to reduce phase conflicts as many as possible and to minimize the total amount of wire segment movement. Our algorithm is designed based on mixed integer linear programming (MILP), and is applicable to both standard-cell and custom layouts. The MILP based al-

gorithm is flexible enough to take other practical issues, such as the lengths of critical nets, into consideration as well. Different from existing works which first solve the phase conflict problem by removing edges from the layout-associated conflict graphs to make the resultant graphs 2-colorable and then try to revise the layout to match the resultant conflict graphs, our algorithm directly fixes the phase conflict problem through wire spreading while at the same time minimizing the total amount of wire segment movement. Compared with the number of edges deleted from conflict graphs, the total amount of wire segment movement is a more direct metric to reflect the cost for revising a layout. Furthermore, the recently remarkable improvement on linear programming solvers [6] has made our algorithm become competitive, and the experimental results well support the effectiveness of our algorithm. The rest of this chapter is organized as follows. In Section 2.1, we describe the preliminaries and our strategy to solve the phase conflict problem using wire spreading. Section 2.2 describes our algorithm. Section 2.3 provides the experimental results which demonstrate the effectiveness of our approach. Finally, Section 2.4 summarizes this chapter.

2.1 Preliminaries and Problem Formulation

2.1.1 Preliminaries

The phase assignment of wire segments on a metal layer can be done by coloring the corresponding conflict graph with two colors. In the graph each node represents a wire segment. If the spacing between a pair of adjacent wire segments falls in $[b, B]$, the two wire segments must be assigned to opposite phases and

we create an edge between their corresponding nodes. The layout is said to be altPSM compliant if and only if all associated conflict graphs are 2-colorable. If two nodes connected by an edge have the same color, we say they incur a phase conflict. For simplicity and clarity, we assume all wire segments are rectangles . The construction of a conflict graph is described as follows. First, we bloat each wire segment by the distance b. Next, we apply the line sweeping algorithm to check if two bloated regions intersect or not. Then, the edges are constructed between pairs of nodes whose corresponding bloated segments intersect. However, according to our empirical study, we observed that if a wire segment is placed at the lower-left or lower-right position of another wire segment and their bloated regions intersect, the phases of these two wire segments can be identical without causing unresolvable aerial image. To demonstrate this observation, we use a layout with four wire segments A, B, C and D as shown in Figure 2.2(a). The dashed rectangles show their bloated regions, and the shaded area highlights the bloated region of C. The four red solid lines and two red dashed lines are the edges of the conflict graph. Figure 2.2(b) shows the simulated aerial image of the four wire segments as well as their phases. The simulation is performed by Silvaco Athena [7], where the wavelength and NA are 193nm and 0.85, respectively, and the wire width and spacing are both 90nm. From the result, we can see that although the bloated regions of B (the upper-left wire segment) and C (the lower-right wire segment) overlap, no image bridging occurs between them and therefore they both can be assigned to the same phase. As a result, we can remove the edge between their corresponding nodes from the conflict graph (*i.e.*, the red

dashed edge connecting B and C in Figure 2.2(a) can be removed). The same observation applies to A (the upper-right wire segment) and D (the lower-left wire segment) as well, and therefore the edge (A, D) in Figure 2.2(a) can be removed. By removing these "crossing" edges, we can guarantee the planarity of the conflict graph. Throughout the rest of this chapter, we will assume conflict graphs are all planar. It is not hard to see that a metal layer is altPSM compliant if and only if its conflict graph does not have any odd face. On the other hand, the minimum number of phase conflicts among all possible phase assignment solutions of a metal layer is bounded by half the number of the odd faces in the corresponding conflict graph, which can be derived from [8]. Therefore, we use the number of odd faces as a metric to measure the amount of phase conflicts. In this chapter, we use wire spreading to remove phase conflicts among wire segments. A proper wire spreading solution induces edge deletion to remove odd faces of a conflict graph. Thus, our goal is to use the wire spreading technique to reduce the number of odd faces as many as possible. For an easier presentation, we only focus on wire spreading along the vertical direction throughout the rest of this chapter, unless stated otherwise. In order to maintain the correct circuit functionality and avoid over distortion for a given layout, the following constraints must be satisfied after wire spreading:

1. **Die region constraint:**

   Since the dimensions of a die have been specified in the very beginning of a physical design flow, it is costly to fix phase conflicts by relaxing the die size. Thus, all the wire segments must locate within the original die region.

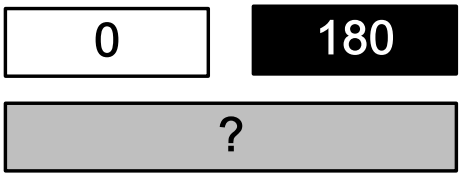


Figure 2.1: A layout which has the phase conflict problem

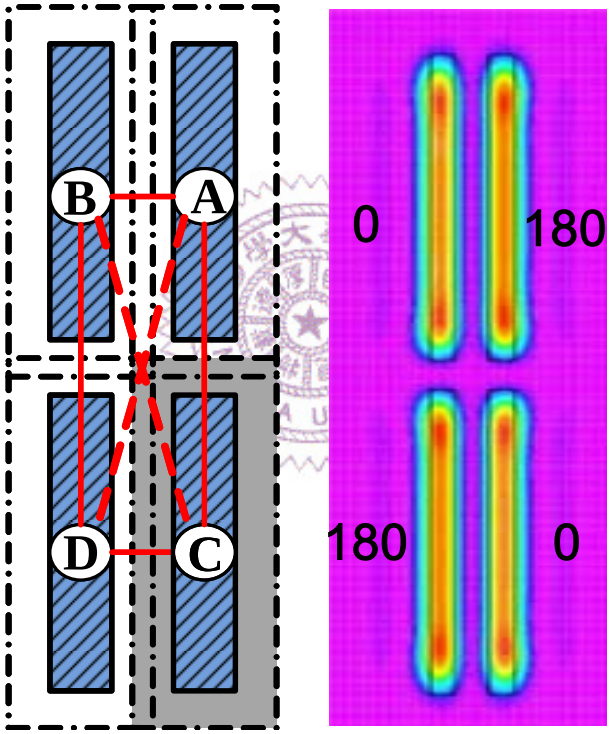


Figure 2.2: (a) The conflict graph of four wire segments. (b) The simulated aerial image.

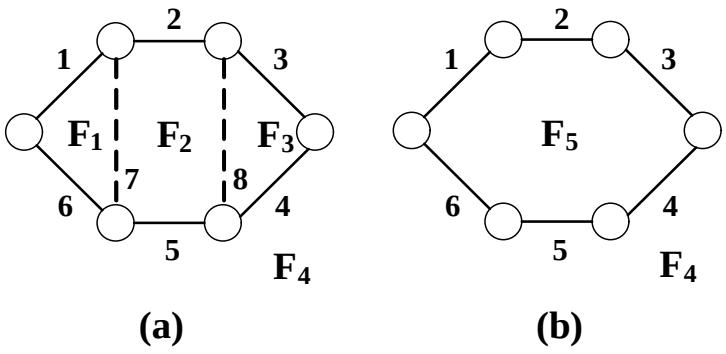


Figure 2.3: Odd face elimination method.

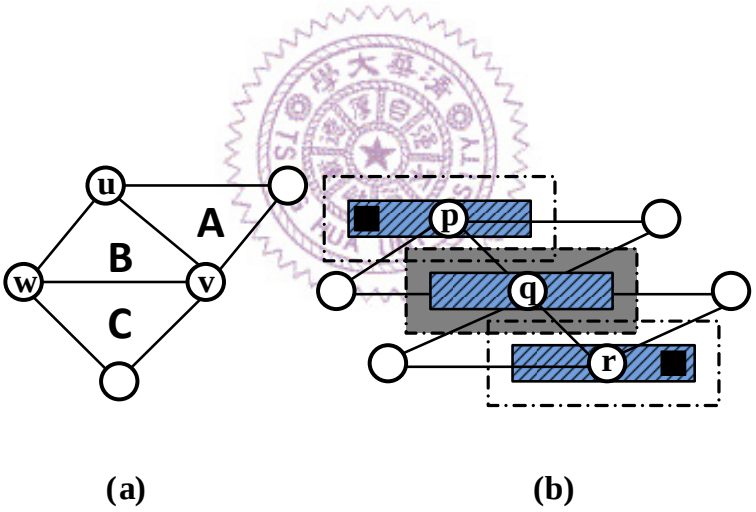


Figure 2.4: (a) A case which overestimates the number of eliminated odd faces. (b) A layout infeasible case.

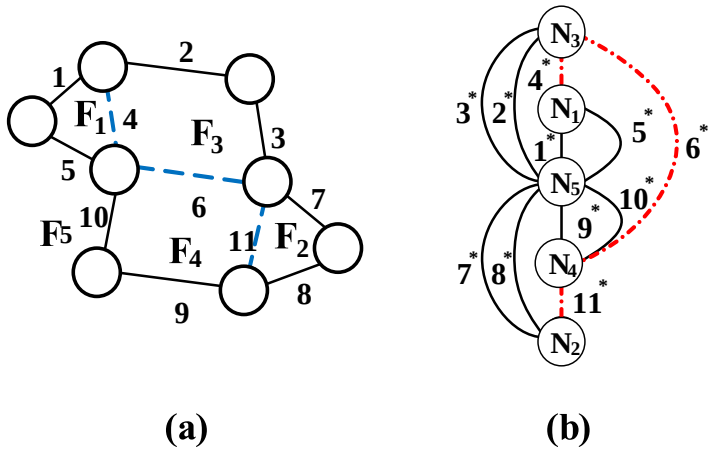


Figure 2.5: An illustration of a conflict graph and its dual graph.

2. **Fixed pin constraint:**

   Since we do not alter the placement results, the I/O pins of cells or hard macros are thought to be immovable. If one end of a wire segment connects with an immovable pin, then the coordinate of this end must be left unchanged.

3. **Connection constraint:**

   If wire segments in different layers are connected through vias, then these connections must be maintained.

4. **Vertical order constraint:**

   To ensure that routing patterns will not be drastically changed, if the horizontal spans of two wire segments in the same layer overlap, then their relative positions in the vertical direction must be held.

5. **Minimum spacing constraint:**

   The spacing between any two wire segments cannot be smaller than minimum spacing b.

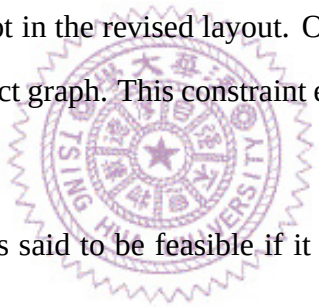
6. **Maximum movement constraint:**

   Each wire segment can be moved only within a user defined range.

   Besides, it is favorable to size down the conflict graphs and not to over modify the original layout. Therefore, the following constraint should also be satisfied after wire spreading:

7. **Graph simplicity constraint:**

   If the distance between two wire segments is longer than B, then this spatial relation should be kept in the revised layout. Otherwise, it will introduce a new edge in the conflict graph. This constraint ensures not to add edges into conflict graphs.



A wire spreading solution is said to be feasible if it meets the above seven constraints.

2.1.2 Problem Formulation

The problem of applying wire spreading to remove phase conflicts is stated as follows: Given a layout which contains n metal layers, we first construct a conflict graph for each of the n layers. The problem asks for finding a new arrangement for the wire segments lying in each of the n layers such that the sum of the numbers of odd faces of the resultant conflict graphs is minimized and the total amount

of wire segment movement is also minimized. In addition, the wire spreading solution must satisfy all the constraints stated in Section 2.1.1. To reduce the number of odd faces in a conflict graph, we need to delete edges from the graph. Each edge in a conflict graph can be exactly categorized into one of the following four types:

1. **Shared by two odd faces:**

   If the edge is deleted, it will merge two odd faces into one even face and therefore the number of odd faces can be reduced by two.

2. **Shared by two even faces:**

   If the edge is deleted, it will merge two even faces into one even face and therefore the number of odd faces will be unchanged.

3. **Shared by one odd face and one even face:**

   If the edge is deleted, it will merge the two faces into one odd face and therefore the number of odd faces will be unchanged.

4. **Not shared by two faces:** If the edge is deleted, it does not merge any face and therefore the number of odd faces will be unchanged.

From the above observations, we have the following lemma.

Lemma 1 *For each edge deletion, the number of odd faces can be reduced if and only if the deleted edge is shared by two odd faces.*

Although merging an odd face with an even face by deleting one of their common edges does not immediately reduce the number of odd faces, it makes the odd

face grow. After several iterations, this growing odd face may directly abut with another odd face and then we can delete one of their common edges to merge them into an even face. The series of mergings can be regarded as pairing two odd faces through a series of even faces. Figure 2.3 shows an example of how to pair two odd faces which are not directly abutted. Initially, two odd faces F_1 and F_3 are not abutted as seen in (a). By sequentially removing edges 7 and 8 from the graph, F_1, F_2 and F_3 gradually merge into an even face F_5 which is shown in (b). In this chapter, we aim to reduce the number of odd faces by pairing odd faces (which may or may not be directly abutted at the beginning). In the following, we define a special edge set called *merging set*.

Definition 1 ***(Merging set):*** *Given a conflict graph, a merging set is a set of edges whose deletion from the graph can merge two odd faces into one even face. For a pair of odd faces, there may exist more than one merging set. A merging set is said to be minimal if there exists no edge e in the set such that after deleting the edge e, the resultant set is still a merging set.*

According to the above definition and *Lemma 1*, deleting all edges in a minimal merging set from a conflict graph reduces the number of odd faces by 2. It is also worth noting that a set of odd face pairings may not always reduce the number of odd faces by twice the number of pairings, or may not be always achievable by wire spreading. We use Figure 2.4(a) and (b) to explain these two cases, respectively. In Figure 2.4(a), we have the merging sets $\{(u, v)\}$ and $\{(v, w)\}$ for odd face pairs (A, B) and (B, C), respectively. Deleting edge (u, v) or edge (v, w) alone merges faces A and B, or B and C into one even face. Intuitively, deleting

two merging sets simultaneously implies to eliminate two pairs of odd faces, and the number of odd faces should be reduced by four. However, because the two odd face pairs share a common face B, simultaneously deleting (u, v) and (v, w) merely merges the three odd faces into one odd face and thus the number of odd faces is reduced only by two. Figure 2.4(b) shows an example where simultaneously performing two odd face pairings may be infeasible. According to the conflict graph shown in Figure 2.4(b), both odd face pairs in the upper and the lower parts can be merged individually by deleting edges (p, q) and (q, r), respectively. However, the wire segments p and r are connected with immovable pins indicated by black squares, and therefore p and r cannot be moved through wire spreading. Now, if the vertical moving range for q (*i.e.*, the range between the bottom boundary of p and the top boundary of r) is not large enough to move q to a location such that both edges (p, q) and (q, r) can be removed from the conflict graph, then simultaneously performing two odd face pairings in this graph becomes impossible.

We next define another special edge set called orthogonal set.

Definition 2 ***(Orthogonal set):*** *Given the conflict graphs corresponding to a layout, an orthogonal set is a union of a collection of minimal merging sets and satisfies the following two conditions:*

1. *In this set, edges belonging to different minimal merging sets do not share any common face.*

2. *There exists at least a feasible wire spreading solution such that all the*

edges in this set are no longer in the new conflict graphs. The cardinality of an orthogonal set is defined to be the number of merging sets involved in the set, which is also equal to half the amount of eliminated odd faces after removing from the conflict graphs all the edges in the set.

Definition 3 ***(Deviation):*** *The deviation induced by an orthogonal set is the minimum total amount of wire segment movement among all feasible wire spreading solutions corresponding to this orthogonal set.*

In this chapter, instead of solving the general wire spreading problem as described at the beginning of this subsection, we only focus on the *multi-layer odd face pairing problem* which asks to find an orthogonal set with maximum cardinality and a corresponding wire spreading solution with the minimum deviation.

2.2 Algorithm

In this section, we first introduce another special edge set called candidate removal set and show the relation between a candidate removal set and an orthogonal set. We then derive an MILP model to find a candidate removal set (and thus an orthogonal set as well) and a corresponding wire spreading solution such that the total cardinality and the total deviation are both optimized.

2.2.1 Candidate Removal Set

Candidate removal set is defined as follows:

Definition 4 ***(Candidate removal set):*** *Assume that we are given n conflict graphs $G_i = (V_i, E_i)$'s, $1 \leq i \leq n$, corresponding to a n-layer layout. Let $G = (V, E)$, where $V = V_1 \bigcup V_2 ... \bigcup V_n$, and $E = E_1 \bigcup E_2 ... \bigcup E_n$. A candidate removal set E' is a subset of E such that there exists at least a feasible wire spreading solution to remove all the edges in E' from G. In addition, E' must satisfy the following constraints:*

1. *For every odd face OF in G, either there is exactly one edge in E' which is also in OF, or no edge in E' is also in OF.*

2. *For every even face EF in G, either there are exactly two edges in E' which are also in EF, or no edge in E' is also in EF.*

In the remaining subsection, we step-by-step explain the relation between a candidate removal set and an orthogonal set. We first have the following lemma which can be proved based on *Lemma 1, Definition 1* and the concept of graph duality.

Lemma 2 *Given a conflict graph G_i and its dual graph D_i, a set M of edges is a minimal merging set in G_i if and only if the set M^* of the dual edges of M forms a path in D_i such that each of the two end nodes of the path corresponds to an odd face in G_i and each of the other nodes in the path corresponds to an even face in G_i.*

We use Figure 2.5 to facilitate the understanding of the above lemma. Figure 2.5(a) depicts a conflict graph which includes 8 nodes, 11 edges and 5 faces. The faces

are denoted by F_1 to F_5, where F_1 and F_2 are the only two odd faces. A minimal merging set M to merge F_1 and F_2 into an even face consists of the edges 4, 6 and 11. Figure 2.5(b) illustrates the dual graph. The nodes N_1 to N_5 are the corresponding nodes of faces F_1 to F_5, respectively. Same tag number is used for each edge and its dual edge, except a superscript asterisk is used to distinguish a dual-graph edge and a conflict-graph edge. The edges 4^*, 6^* and 11^* which are dual edges of M form a path N_1-N_3-N_4-N_2 in the dual graph. The two end nodes N_1 and N_2 correspond to the odd faces F_1 and F_2 while the other nodes N_3 and N_4 correspond to the even faces F_3 and F_4.

Lemma 3 *If two minimal merging sets of a conflict graph are involved in the same orthogonal set, their corresponding paths in the dual graph are node-disjoint.*

Proof. *By* Lemma 2*, the dual edges of each minimal merging set form a path. By* Definition 2*, the two minimal merging sets do not share any common face. Thus, the corresponding paths of the two merging sets are node-disjoint in the dual graph.* □

Theorem 1 ***(Orthogonal set inclusion theorem):*** *Any candidate removal set contains an orthogonal set.*

Proof. *According to the concept of graph duality and the constraints (1) and (2) imposed on a candidate removal set, the dual edges of a candidate removal set E' can only compose two kinds of connected components-paths and cycles. We can further uniquely decompose E' into two disjoint edge sets P and C, and use P^* and C^* to denote their dual edge sets, respectively, where P^* is composed of a set*

of node-disjoint paths and C^* is composed of a set of cycles. Moreover, the two end nodes of each path in P^* correspond to odd faces in G and the other nodes in the path correspond to even faces in G. Therefore, according to Lemma 2 *and* Lemma 3, *P is a collection of minimal merging sets which do not share common faces in G. Besides, if S is a feasible wire spreading solution for E', then S is also a feasible wire spreading solution for P, according to* Definitions 2 *and* 4. *As a result, P is an orthogonal set.* □

It is not hard to see that in the proof of *Theorem 1*, the number of paths in P^* is half the number of odd-degree nodes in P^*. Thus, the cardinality of P is half the number of odd-degree nodes in P^*. For convenience, we define the equivalent cardinality of the candidate removal set E' to be the number of paths in P^*. Based on *Theorem 1*, the multi-layer odd face pairing problem will now be tackled by solving the following problem.

Candidate removal set problem: Given an n-layer layout, this problem asks for finding a candidate removal set and a corresponding feasible wire spreading solution such that the equivalent cardinality of the candidate removal set is maximized and the induced deviation is minimized.

2.2.2 An MILP-Based Wire Spreading Algorithm

Table 2.1 lists the notations and the definitions of the variables to be used in the description of our MILP based algorithm. First, we construct a conflict graph Gi for the ith layer of the layout. Each node $N_{i,j}$ is represented by a five tulpe $(x_{0,i,j}, y_{0,i,j}, x_{1,i,j}, y_{1,i,j}, w_{i,j})$ which records the geometric information of the wire

segment of $N_{i,j}$. Then, the candidate removal set problem is formulated as an MILP model below.

Maximize $\beta \sum_{i=1}^{n} \sum_{l=1}^{o_i} (f_{i,l} | F_{i,l} \; is \; an \; odd \; face) - \sum_{i=1}^{n} \sum_{j=1}^{m_i} (\sigma_{0,i,j} + \sigma_{1,i,j} + \epsilon_{0,i,j} + \epsilon_{1,i,j})$

subject to

$$y_{0,i,j} - w_{i,j} \geq 0, \forall i, j : 1 \leq i \leq n, 1 \leq j \leq m \quad (2.1)$$

$$y_{0,i,j} + w_{i,j} \leq H, \forall i, j : 1 \leq i \leq n, 1 \leq j \leq m \quad (2.2)$$

$$y_{0,i,j} = y^*_{0,i,j}, \forall i, j : 1 \leq i \leq n, 1 \leq j \leq m, T_{0,i,j} \; connects \; with \; a \; fixed \; pin \quad (2.3)$$

$$y_{1,i,j} = y^*_{1,i,j}, \forall i, j : 1 \leq i \leq n, 1 \leq j \leq m, T_{1,i,j} \; connects \; with \; a \; fixed \; pin \quad (2.4)$$

$$y^*_{0,i,j} - movement \leq y_{0,i,j} \leq y^*_{0,i,j} + movement, \forall i, j : 1 \leq i \leq n, 1 \leq j \leq m_i \quad (2.5)$$

$$y^*_{1,i,j} - movement \leq y_{1,i,j} \leq y^*_{1,i,j} + movement, \forall i, j : 1 \leq i \leq n, 1 \leq j \leq m_i \quad (2.6)$$

$$y_{0,i,j} = y_{0,i+1,j}, \forall i, j, k : 1 \leq i \leq n-1, 1 \leq j \leq m_i, 1 \leq k \leq m_{i+1}, T_{0,i,j} \; connects \; T_{0,i+1,k} \quad (2.7)$$

$$y_{0,i,j} - w_{i,j} - w_{i,k} - y_{1,i,k} \geq b, \forall i, j, k : 1 \leq i \leq n, 1 \leq j \leq m_i, j < k \leq m_i, y_{0,i,j} \geq y_{1,i,k},$$
$$the \; horizontal \; spans \; of \; the \; bloated \; regions \; of \; N_{i,j} \; and \; N_{i,k} \; overlap \quad (2.8)$$

$$y_{0,i,j} - wi, j - w_{i,k} - y_{1,i,k} \geq B, \forall i, j, k : 1 \leq i \leq n, 1 \leq j \leq m_i,$$
$$j < k \leq m_i, E_{i,j,k} \notin G_i, y^*_{0,i,j} \geq y^*_{1,i,k} \quad (2.9)$$

$$y_{0,i,j} = y_{1,i,j}, \forall i, j : 1 \leq i \leq n, 1 \leq j \leq m_i, N_{i,j} \; is \; a \; horizontal \; wire \; segment \quad (2.10)$$

$$y_{0,i,j} - wi,j - w_{i,k} - y_{1,i,k} \geq B \times e_{i,j,k}, \forall i,j,k : 1 \leq i \leq n, 1 \leq j \leq m_i,$$

$$j < k \leq m_i, E_{i,j,k} \in G_i, y^*_{0,i,j} \geq y^*_{1,i,k} \quad (2.11)$$

$$f_{i,l} = \sum_{j=1}^{m_i} \sum_{k=j}^{m_i} (e_{i,j,k} | E_{i,j,k} \in F_{i,l} \; and \; F_{i,l} \; is \; an \; odd \; face) \forall i,l : 1 \leq i \leq n, 1 \leq l \leq o_i \quad (2.12)$$

$$2 \times f_{i,l} = \sum_{j=1}^{m_i} \sum_{k=j}^{m_i} (e_{i,j,k} | E_{i,j,k} \in F_{i,l} \; and \; F_{i,l} \; is \; an \; even \; face) \forall i,l : 1 \leq i \leq n, 1 \leq l \leq o_i \quad (2.13)$$

$$\sigma_{0,i,j} \geq 0, \; and \; y^*_{0,i,j} - y_{0,i,j} + \sigma_{0,i,j} \geq, \forall i,j : 1 \leq i \leq n, 1 \leq j \leq m_i \quad (2.14)$$

$$\epsilon_{0,i,j} \geq 0, \; and \; y_{0,i,j} - y^*_{0,i,j} + \epsilon_{0,i,j} \geq, \forall i,j : 1 \leq i \leq n, 1 \leq j \leq m_i \quad (2.15)$$

$$\sigma_{1,i,j} \geq 0, \; and \; y^*_{1,i,j} - y_{1,i,j} + \sigma_{1,i,j} \geq, \forall i,j : 1 \leq i \leq n, 1 \leq j \leq m_i \quad (2.16)$$

$$\epsilon_{1,i,j} \geq 0, \; and \; y_{1,i,j} - y^*_{1,i,j} + \epsilon_{1,i,j} \geq 0, \forall i,j : 1 \leq i \leq n, 1 \leq j \leq m_i \quad (2.17)$$

$$x_{0,i,j}, y_{0,i,j}, x_{1,i,j}, y_{1,i,j}, \sigma_{0,i,j}, \sigma_{1,i,j}, \epsilon_{0,i,j}, \epsilon_{1,i,j} \in \mathbb{R}, \forall i,j : 1 \leq i \leq n, 1 \leq j \leq m_i \quad (2.18)$$

$$e_{i,j,k} \in \{0,1\}, \forall i,j,k : 1 \leq i \leq n, 1 \leq j \leq m_i, j \leq k \leq m_i, E_{i,j,k} \in G_i \quad (2.19)$$

$$f_{i,l} \in \{0,1\}, \forall i,l : 1 \leq i \leq n, 1 \leq l \leq o_i \quad (2.20)$$

The objective function can be decomposed into two terms. The first term is the gain function which is equal to the reduction of number of odd faces; i.e. twice the cardinality of a candidate removal set. The second term is the penalty function which is equal to the deviation induced by revising the layout. A constant β is introduced to provide a tradeoff between these two terms. Constraints (1) and (2) require wire segments to locate within a fixed die area. Constraints (3) and (4) bind the ends of wire segments with fixed pins. Constraints (5) and (6) confine wire segments to move within the given allowable range. Constraint (7)

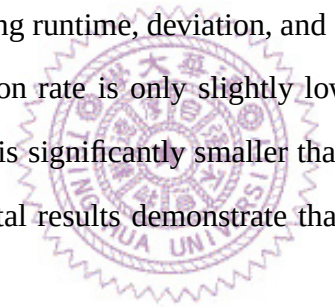
maintains the connectivity among wire segments. To satisfy the minimum spacing rule of altPSM and to keep the relative vertical order of wire segments, we have constraint (8). To maintain the orientations of horizontal wire segments, we have constraint (9). To preserve the graph simplicity constraint, we have constraint (10). In constraint (11), we use a Boolean variable $e_{i,j,k}$ to determine whether edge $E_{i,j,k}$ is removed or not. If $e_{i,j,k}$ is assigned to 1, $E_{i,j,k}$ is removed from the corresponding conflict graph. As a consequence, the spacing between wire segments $N_{i,j}$ and $N_{i,k}$ must be greater than B. To satisfy the constraints imposed on a candidate removal set, we have constraints (12) and (13). To calculate the deviation of both ends of a wire segment, we apply a relaxation technique. With constraints (14) and (15), the sum of $\sigma_{0,i,j}$ and $\epsilon_{0,i,j}$ is exactly the absolute value of difference between $y_{0,i,j}$ and $y^*_{0,i,j}$, when we minimize $\sigma_{0,i,j}$ and $\epsilon_{0,i,j}$. Similar principles are applied to calculate the deviation of $T_{1,i,j}$. Thus we have constraints (16) and (17). Finally, the value ranges of all variables are given by constraints (18), (19) and (20). In this MILP model, we introduce six real variables for each node, one binary variable for each edge and one binary variable for each face. Assume that the number of nodes, edges and faces are v, e and f, respectively. The total number of variables is $6v + e + f$. In the given planar graphs, both e and f are of the same order as v. Thus, the number of variables is $O(v)$. Obviously we can see that, except constraint (8) whose number grows with v^2, the number of the rest of the constraints linearly grows with v, e or f. Therefore, the number of constraints is $O(v^2)$.

2.3 Experiment Results

Our algorithm was implemented in C++ and run on an Intel 1.6GHz Linux machine. We used CPLEX [9] to solve the MILP problems. The values of B and b are 160nm and 80nm, respectively. We adopted randomly generated layouts with the same minimum spacing and wire width to test the robustness and stability of our algorithm. All the layouts use 4 metal layers. Table 2.2 shows the statistics of each layout. The names of layouts are listed in the first column. The *#node*, *#edge* and *#odd_face* respectively show the numbers of nodes, edges and odd faces of the conflict graphs before wire spreading. The number of nets for each layout is indicated in the column *#net*. For general consideration, we assume that some pins are immovable.

The experimental results are summarized in Table 2.3. Besides our algorithm, we also present the results of two other algorithms (the adjacent pairing algorithm and the aggressive algorithm) for comparison. The adjacent pairing algorithm removes the odd faces by deleting edges which are shared by two abutted odd faces, and the aggressive algorithm eliminates odd faces as many as possible without considering the deviation. Both of these two algorithms are derived from our algorithm by either modifying some constraints or the objective function. For each layout, each algorithm performed one run of vertical wire spreading followed by horizontal wire spreading. The columns *#rof*, *deviation*, *%comp* and runtime in Table 2.3 show the number of remaining odd faces, the induced deviation (nm), the reduction rate of odd faces, and the run time (sec), respectively. To make com-

parisons among these algorithms, we normalize the value of each column with respect to the results of our algorithm. The adjacent pairing algorithm intuitively solves the phase conflict problem using the observation of *Lemma 1*. Although it is the fastest algorithm among the three algorithms, the solution space is only a subset of the other two algorithms. Many odd faces which are surrounded by even faces have no chance to merge with other odd faces. Therefore, there are still many odd faces left after wire spreading. On the contrary, the aggressive algorithm completely eliminates all odd faces and the runtime is shorter than our algorithm. However, it induces a large amount of deviation, which may impact the circuit timing. In order to emphasize the importance of deviation, we assigned β a relatively small value in our algorithm. The results indicate that our algorithm shows a good tradeoff among runtime, deviation, and completion rate of odd face elimination. Our completion rate is only slightly lower than the aggressive algorithm, and the deviation is significantly smaller than the aggressive algorithm. In addition, the experimental results demonstrate that the runtime of our MILP algorithm is reasonable.



2.4 Summary

In this chapter, we present an MILP-based wire spreading algorithm to solve the phase conflict problem with minimal perturbation to the given layout. Different from the previous works which ask for finding a minimum weighted edge set or a minimal edge set whose removal makes the conflict graph 2-colorable, our algorithm use the exact deviation of wire segments as cost. The experimental

results show the effectiveness of our algorithm; less than 2% of odd faces are left in the modified layout and no area increase is needed for the modification. In addition, the modification will not alter the lengths of the critical nets, if we add linear constraints to control the critical wire lengths. The flexible MILP model also allows integrating with a variety of practical issues such as redundant via insertion [10].

Table 2.1: Notation list

H	The height of the region
B	The minimum spacing without applying altPSM
b	The achievable minimum spacing of altPSM
$movement$	Maximum allowable vertical movement of wire segments
n	The number of layers
G_i	The conflict graph of the ith layer
E	a set of nets
m_i	The number of nodes in G_i
o_i	The number of faces in G_i
$N_{i,j}$	The jth node in the conflict graph G_i
$E_{i,j,k}$	The edge between $N_{i,j}$ and $N_{i,k}$
$T_{0,i,j}$	If $N_{i,j}$ is a horizontal wire segment, $T_{0,i,j}$ is the left end of the center line of $N_{i,j}$. Otherwise, $T_{0,i,j}$ is the lower end of the center line of $N_{i,j}$.
$T_{0,i,j}$	If $N_{i,j}$ is a horizontal wire segment, $T_{0,i,j}$ is the right end of the center line of $N_{i,j}$. Otherwise, $T_{0,i,j}$ is the upper end of the center line of $N_{i,j}$.
$e_{i,j,k}$	A Boolean variable to control the deletion of edge $E_{i,j,k}$. If $e_{i,j,k}$ is 1, edge $E_{i,j,k}$ is removed from G_i.
$f_{i,l}$	A Boolean variable to determine the number of deleted edge(s) of face $F_{i,l}$. If $F_{i,l}$ is an odd face, we remove one edge from $F_{i,l}$ when $f_{i,l}$ is 1. If $F_{i,l}$ is an even face, we remove two edges from $F_{i,l}$ when $f_{i,l}$ is 1. If $f_{i,l}$ is 0, no edge is removed from from $F_{i,l}$.
$(x_{0,i,j}, y_{0,i,j})$	The coordinate of $T_{0,i,j}$ after wire spreading.
$(x_{1,i,j}, y_{1,i,j})$	The coordinate of $T_{1,i,j}$ after wire spreading.
$(x^*_{0,i,j}, y^*_{0,i,j})$	The original coordinate of $T_{0,i,j}$.
$(x^*_{1,i,j}, y^*_{1,i,j})$	The original coordinate of $T_{1,i,j}$.
$w_{i,j}$	The half width of wire segment$N_{i,j}$.
$\sigma_{0,i,j}$, $\sigma_{1,i,j}$, $\epsilon_{0,i,j}$, $\epsilon_{1,i,j}$	Four relaxation real variables to calculate the deviation of both ends of wire segment $N_{i,j}$.

Table 2.2: Benchmark layout parameters.

	#node	#edge	#odd_face	#net
C1	1009	436	14	254
C2	4984	2088	58	1702
C3	10974	6668	330	3514
C4	19642	12578	668	6175
C5	57182	36373	1908	17787

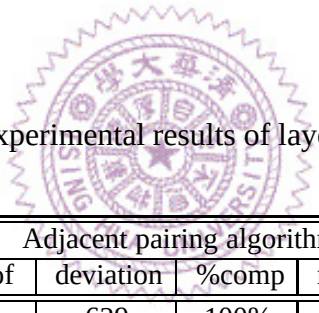


Table 2.3: The experimental results of layout modification.

	Our algorithm				Adjacent pairing algorithm				Aggressive algorithm			
	#rof	deviation	%comp	runtime	#rof	deviation	%comp	runtime	#rof	deviation	%comp	runtime
C1	0	629	100%	0.2	0	629	100%	0.2	0	677424	100%	0.2
C2	0	3667	100%	1.8	0	3503	100%	2.9	0	3504300	100%	1.3
C3	0	18691	99.39%	9.1	48	13599	85.45%	8.2	0	13756230	100%	8.6
C4	0	33967	98.88%	24.0	102	23339	84.73%	9.9	0	23933200	100%	22.1
C5	0	96535	98.43%	297.7	324	70711	83.02%	131.4	0	69980000	100%	140.8
	normalized				normalized				normalized			
	#rof	deviation	%comp	runtime	#rof	deviation	%comp	runtime	#rof	deviation	%comp	runtime
	1	1	1	1	11.85	0.73	0.85	0.46	0	728.7	1.01	0.52

Chapter 3

Through-Silicon Via Planning in 3D Floorplanning

Three dimensional integrated circuit (3D-IC) technology is a promising solution for performance improvement. Compared to a traditional two-dimensional (2D) IC design, which places all the devices on one single planar tier, a 3D-IC stacking of multiple tiers allows more devices to be placed closely, hence substantially reducing wirelength. It has been shown [11] that wirelength can be reduced by 15% in a 3-tier 3D-IC, and 42% in a 4-tier 3D-IC. In addition to performance improvement, 3D-ICs provide many benefits including high density, high bandwidth, and low power [12].

Based on the stacking style, 3D integrations can be classified into three major categories: face-to-face (F2F), face-to-back (F2B), and back-to-back (B2B) chip stacking. A basic F2F stacking process only stacks two chips face-to-face. The inter-chip connections directly go through the joined bonding pads in front sides of both chips. For more than two tiers, F2B or B2B chip stacking is inevitable. F2B chip stacking is the most commonly used multi-chip stacking process. The key to

realizing F2B chip stacking is through-silicon via (TSV) technology. A TSV is a vertical metal channel that directly penetrates through the silicon substrate at any feasible white space and connects an inter-chip net. In this chapter, we will focus on the F2B chip stack.

In spite of tremendous benefits brought by the 3D integration, the 3D integration itself also bring extra burdens to a design. Since a TSV is much larger than an ordinary via, the placement of a TSV is more restricted. It has been pointed out that the bottleneck of net routing is related to signal TSVs [13]. The deployment and feasibility of signal TSVs are crucial to a routing stage. Without considering them in the early design stage, the performance of a design may degrade seriously, or routing cannot even be completed. Therefore, floorplanning tools for 3D-ICs must take signal TSV planning into account. Previous research is reviewed as follows. He *et al.* [15] proposed a buffer and inter-layer via planning algorithm in floorplanning. For each net, the algorithm identifies the bounding box of source and sinks as a feasible region, and tries to place as many buffers and TSVs as possible within the feasible region. However, the white space within the restricted feasible region may not always be sufficient to accommodate buffers and TSVs. Thus, their algorithm cannot guarantee completed routing. In fact, a preliminary experimental result in the next section reveals that a large number of TSVs cannot be put in the bounding boxes. Lu *et al.* [16] proposed a post-floorplan TSV placement algorithm to improve the performance of designs. However, the optimality of their results strongly depends on the given floorplan.

Another research category focusing on the thermal issue of 3D-ICs is intro-

duced as follows. Cong *et al.* [17] proposed a thermal-driven floorplanning algorithm that simultaneously minimizes chip area, wirelength, number of TSVs and temperature. In addition, a 2-stage floorplanning approach is proposed to reduce the design complexity [18]. The first stage arranges modules into different tiers to reduce the number of TSVs. Then, the second stage determines the floorplan of each tier. Since TSV is a good medium for heat dissipation, a thermal TSV planning scheme is integrated into their algorithm. To make the 3D floorplanning more scalable, Zhou *et al.* also proposed a force directed algorithm to optimize the area, wirelength and peak temperature of chips [19]. In addition, Li *et al.* proposed a linear programming based algorithm to redistribute the white space for thermal TSVs insertion [20]. To cope with the thermal problem, several placers [21–23] simultaneously optimize the temperature and the wirelength in the placement stage. In addition, thermal-driven routers which place thermal TSVs for temperature reduction are proposed in many papers [24] [25].

In this chapter, we will study floorplanning in 3D-IC. Although literature is abundant on 3D-IC floorplanning, none of them consider the areas and positions of signal TSVs. However, the stress induced by TSV impacts the performance of neighboring device. Thus, a large keep out zone (KOZ) is conserved in the neighboring area of a TSV and limits the size of TSV.In previous research, signal TSVs are viewed only as points during floorplanning stage. Ignoring the areas, positions and connections of signal TSVs, previous research estimates the wirelength by measuring the bounding box of pins in a net only. Therefore, in this chapter, we propose a 3D floorplaning algorithm that plans TSVs for wirelength

reduction.

The rest of the chapter is organized as follows. Section 3.1 presents the motivation of this work. Section 3.2 describes details of the F2B stacking technology and defines our problem. In Section 3.3, the details of our algorithm are illustrated step by step. The experimental results are shown and discussed in Section 3.4. Finally, the summary of this chapter is presented in Section 3.5. Since we focus on signal TSV planning, throughout the rest of this chapter, the term TSV refers to signal TSV, unless otherwise stated.

3.1 Motivation

With the advance of 3D integration technology, TSV technology has become a reliable technique for realizing inter-tier connections. However, previous research on 3D-IC floorplanning does not consider the placement problem of signal TSVs [17] [19]. Signal TSVs are viewed as points during floorplanning stage and placed in white space after floorplanning is performed. However, even after applying the leading-edge fabrication process, the size of TSV is still larger than that of ordinary vias by two orders. Consider a TSV with a size of $10\mu m$ by $10\mu m$, the area of TSV is 80x larger than that of a basic cell (about $1.2\mu m^2$ [26]) in a 65nm technology. In a 3D-IC, a thousand of TSVs will take up an area of $0.1mm^2$, which is comparable to the size of a computation core. Thus, we cannot ignore the size of TSV in the early design stage, and assume the signal TSVs can always be arranged in any position in layouts.

According to our observations, the wirelength of a net changes significantly

with the positions of its TSVs. Figure 3.1 shows how the position of a TSV affects the wirelength of a floorplanning result. For simplicity, we assume there are only one net and five blocks on two tiers. The net is an inter-tier net that connects pins S and T. By swapping blocks 1 and 2 of floorplan a, we produce floorplan b. Without considering the positions of TSVs, one may directly estimate the wirelength by measuring the half-perimeter length of the bounding box of its pins, S and T. The bold rectangles in the figure indicate bounding boxes of the net pins in two different floorplans. Since the bounding box in floorplan a is smaller than that in floorplan b, floorplan a seems like a better solution. However, in floorplan a, there is no white space within the bounding box. To connect this inter-tier net, we have to place the TSV in a nearby white space on tier 2, thus the wirelength significantly increases. On the other hand, in floorplan b, there is available white space for a TSV within the bounding box. Therefore, the net can be routed by the shortest path. As is shown, the HPWL (half-perimeter wirelength) without considering TSVs underestimates the wirelength of 3D nets.

Next, we conducted a preliminary experiment to understand how the total wirelength is underestimated when assuming signal TSVs are always put in the white space within the bounding box of pins. We used *ami33* in the MCNC benchmark suite as a study case. The size of TSV was set to be $20\mu m$ by $20\mu m$ [14], and the number of tiers was 3. We modified a 2D floorplanner, Parquet [27], to a 3D floorplanner, and used the modified algorithm to generate some 3D floorplans. Since previous researchers used 15% [27] [28] of white space for 2D fixed-outline floorplanning problems, we also used 15% of white space in the experiment. Fig-

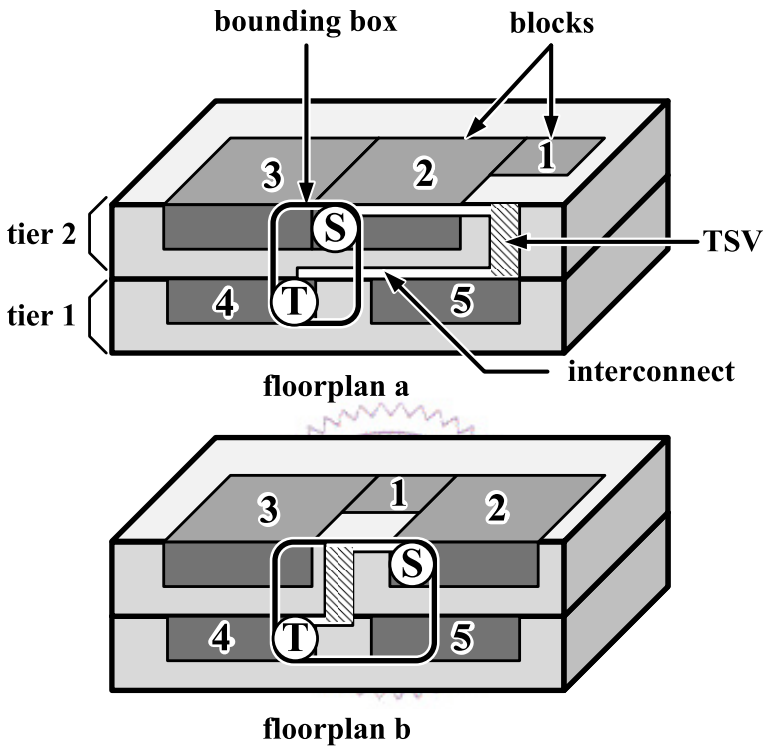


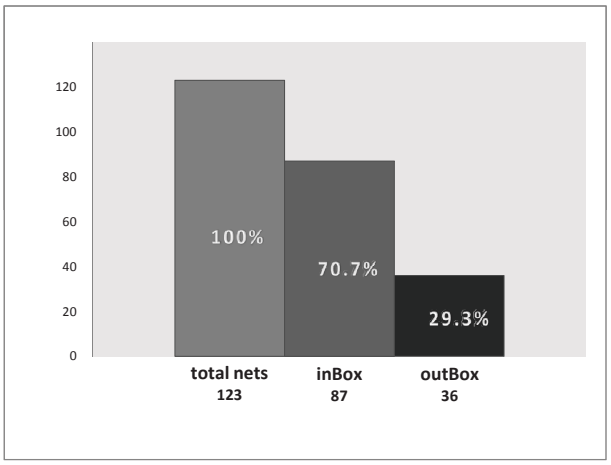
Figure 3.1: A demonstration of how TSV position affects wirelength.

ure 3.2(a) shows the statistical results of numbers of nets. The *total nets* shows the total number of nets of the benchmark circuit. The *inBox* indicates the number of nets whose signal TSVs are totally placed into the white space within the bounding box of net pins. The *outBox* is the number of nets with signal TSVs outside the bounding box of net pins. As the results demonstrate, 29.3% of nets possess TSVs that cannot be put into the white space within the bounding box of pins. Figure 3.2(b) shows results of the wirelengths. The *total HPWL* is the sum of net HPWLs that consider both pins and signal TSVs of nets. The *pin HPWL* is the sum of HPWLs of net pins. The *error* shows the difference between wirelengths with and without considering TSV. It shows that the total wirelength is underestimated by 26.8%.

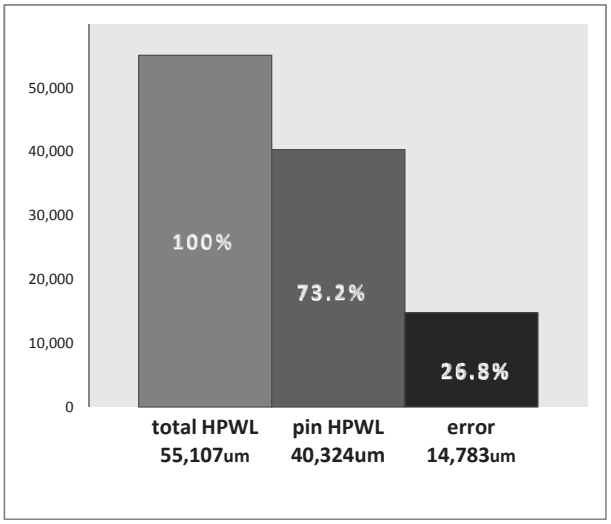
With the above-mentioned observations, we find that it is important to consider both the positions and the placement feasibility of signal TSVs in the floorplaning stage. Although Li *et al.* [18] consider the the problem of TSV planning, they aim to alleviate the thermal impact of 3D-ICs by planning thermal TSVs. Therefore, the deployment of TSVs is not necessarily in favor of wirelength optimization. To fully exploit 3D-IC technology, we must to develop a floorplanning algorithm that plans the signal TSVs while taking into account wirelength minimization and feasibility of TSV placement.

3.2 Problem Formulation and Modeling

In this section, we will first comment on the most commonly used 3D integration technology, the F2B integration technology, in Section 3.2.1. Next, a list of nota-



(a) The numbers of nets which are put within/without the bounding boxes of net pins



(b) Difference between HPWLs with and without considering signal TSVs

Figure 3.2: Statistical results of numbers of nets.

tions used throughout this chapter is presented in Table 3.1. Then, the wirelength estimation method is described in Section 3.2.2. Finally, the problem formulation is presented in Section 3.2.4. Though thermal issue is not the major discussion of our work, it is still a problem worth to be concerned. Therefore, in the last of the section, we briefly describe a fast thermal analysis which can be easily integrated into our algorithm to cope with the thermal problem.

3.2.1 Face-to-Back Integration Technology

Figure 3.3 is an illustration of an F2B chip stacking with three tiers. Each tier is fabricated individually and thinned down by a backside grinding process. The chips are then precisely aligned and bound together by epoxy. Since the diameter of TSVs can be smaller than the accuracy of alignment, a minor misalignment will fail the connections among TSVs. To assure the connections, large landing pads are designed on the top metal layer. The size of a landing pad is about the size of TSV plus the alignment accuracy [29]. For example, if the size of a TSV is $5\mu m$ and the alignment accuracy is also $5\mu m$, the size of a landing pad would be $10\mu m$. As long as TSVs land on the corresponding landing pads, metallic bonding points can form.

Consider a hypothetical signal that goes from pin p to pin q. An inter-tier net must go through the device layer of tier 3 to the device layer of tier 1. Thus, one signal TSV is required in tier 3 and one in tier 2. However, the signal does not pass through the substrate of the lowest tier of this net. Hence, a TSV in tier 1 is not required.

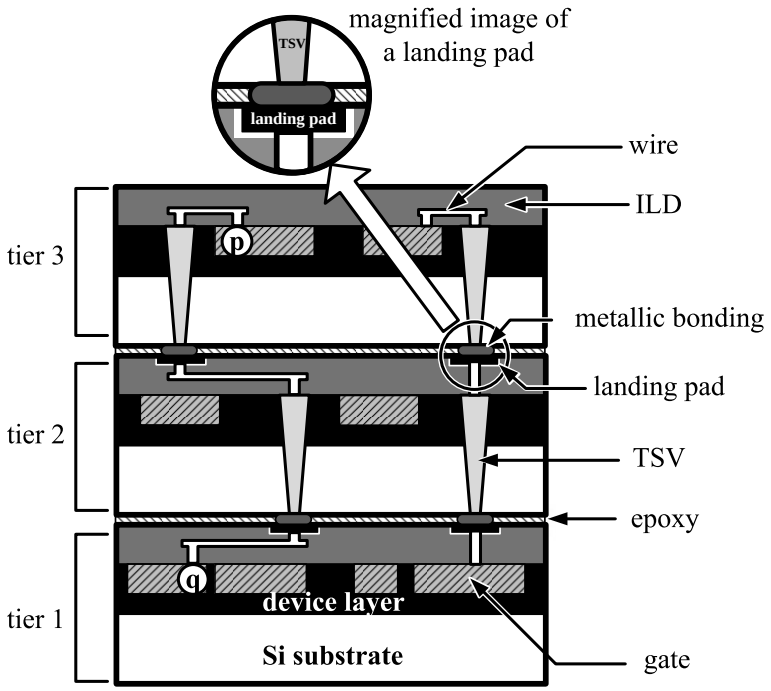
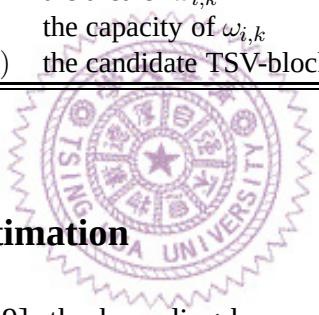


Figure 3.3: Face-to-back chip stacking.

Table 3.1: Notation list

k	the number of tiers
B	a set of blocks
b_i	the i-th block of B
(x_i, y_i, z_i)	the coordinate of lower-left corner of b_i, where x_i, y_i are real numbers and z_i is an integer
E	a set of nets
e_i	the i-th net of E
w_i, h_i	the width and height of b_i
w_{tier}, h_{tier}	the width and height of a tier
w_{TSV}, h_{TSV}, a_{TSV}	the width, height and area of a TSV
$\tau_{i,j}$	the TSV of e_i in the j-th tier
$\omega_{i,k}$	the k-th TSV-block in the i-th tier
$width(\omega_{i,k})$, $height(\omega_{i,k})$	the width and height of $\omega_{i,k}$
Ω_i	the set of the TSV-blocks in the i-th tier
$area(\omega_{i,k})$	the area of $\omega_{i,k}$
$cap(\omega_{i,k})$	the capacity of $\omega_{i,k}$
$candidates(\tau_{i,j})$	the candidate TSV-blocks of $\tau_{i,j}$



3.2.2 Wirelength Estimation

In the previous work [17–19], the bounding boxes of nets in 3D are used as a metric for wirelength estimation. However, as mentioned in the last section, the placement of TSVs greatly affects the wirelength of a 3D-IC. Ignoring the locations of TSVs, traditional HPWL metric fails in estimating the wirelength of 3D floorplans. To facilitate TSV placement, we cluster TSVs into TSV-blocks. Consider an inter-tier net e_i spans from tier u to tier v, where $u < v$. For each tier j crossed by e_i, we assign a TSV, $\tau_{i,j}$, to an appropriate TSV block (where

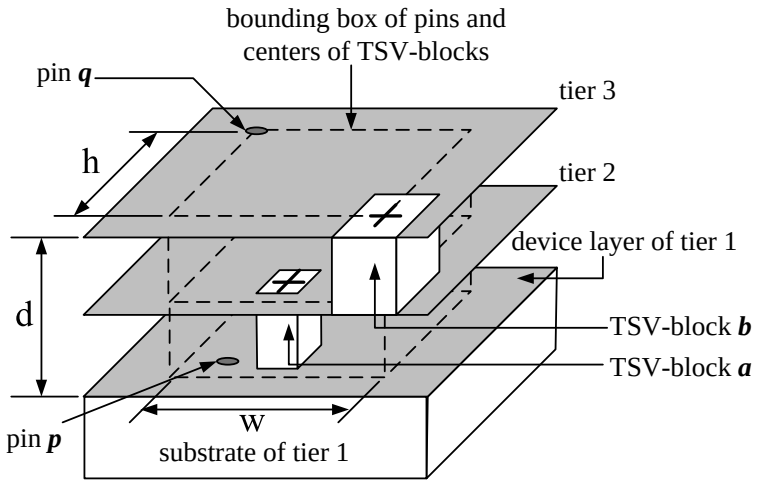


Figure 3.4: Our wirelength estimation method

$u + 1 \leq j \leq v$, please see Section 3.2.1). We define the wirelength of net e_i as the sum of the width, height and depth of the bounding box of *its pins and the centers of all TSV blocks* $\omega_{j,k}$*'s that TSV* $\tau_{i,j}$*'s are assigned to*. Figure 3.4 shows how we estimate the wirelength of a 3D net. To demonstrate the underlying pins and TSV blocks clearly, we remove the substrates of tier 2 and 3, and only display the substrate of tier 1. As can be seen, there is a two-pin net whose two terminals p and q locate at tier 1 and tier 3, respectively. The TSVs of this net are placed in TSV-blocks a and b whose centers are marked by cross marks. The dashed cube is the 3D bounding box of all the pins and the centers of TSV-blocks of this net. The wirelength is defined as the sum of w, h and d. For a 2D net, the wirelength is defined as the half-perimeter length of the bounding box of pins.

3.2.3 Fixed-Outline Floorplan

Since TSVs must land precisely on corresponding landing pads (see Figure 3.3), the die alignment process becomes crucial to the 3D integration. Different bonding processes have different alignment accuracies and different requirements on the sizes of stacked dies. Thus, in the following, we will classify the bonding processes into two categories and discuss their requirements on sizes of stacked dies. The first category is the wafer-to-wafer (W2W) bonding process. This process first aligns the stacked wafers and bonds them together layer by layer, then slices bonded wafers to individual 3D-ICs. The W2W bonding has a high alignment accuracy ($\simeq 1\mu m$ [14]). Therefore, it can achieve a high quality bonding. However, the sizes of stacked dies must be identical. The second category is die-to-die (D2D)/die-to-wafer (D2W) bonding. In these two processes, sliced dies are aligned with their carrier dies/wafers and then bonded with the carrier dies/wafers one by one. Although dies with different sizes are allowed to be stacked together, large differences among die sizes will increase the difficulty of bonding. The alignment accuracy ($5 \sim 15\mu m$ [14]) of D2D/D2W bonding is lower than that of W2W bonding, which make the alignment sensitive to the differences of die sizes. Additionally, in a chip stack, both width and height of a die on top cannot be larger than those of an underlying die. Therefore, the shapes of dies cannot change arbitrarily.

From the previous description, it is beneficial to define the shape before floorplanning to take full advantage of 3D integration for yield enhancement. Hence, in this chapter, we will focus on a fixed-outline floorplanning.

3.2.4 Problem Formulation

Given the following inputs: (1) a block set B, where each block b_i in B has a fixed width and height, w_i and h_i, (2) a netlist E, (3) the number of device tiers k, (4) the width and height of a tier, w_{tier} and h_{tier}, and (5) the width and height of a TSV, w_{TSV} and h_{TSV}, our goal is to determine the values of the following variables:

1. The coordinate (x_i, y_i, z_i) of each block b_i.
2. The coordinates and the sizes of all TSV-blocks.
3. For each TSV $\tau_{i,j}$,i.e., the TSV of e_i on the j-th tier, a TSV-block $\omega_{j,k}$ that $\tau_{i,j}$ is assigned to.

Meanwhile, the wirelength is minimized, and the following constraints are satisfied.

1. $0 \leq x_i \leq w_{tier} - w_i, 0 \leq y_i \leq h_{tier} - h_i$ and $1 \leq z_i \leq k$.
2. No two blocks overlap.
3. For each TSV-block $\omega_{i,k}$, the area of $\omega_{i,k}$ is no less than the sum of the areas of all TSVs assigned to $\omega_{i,k}$.

3.2.5 Thermal Analysis

Since we focus on the TSV planning issue, a fast thermal analysis is required in our floorplanning algorithm. Thus, we adopt a simplified thermal model according

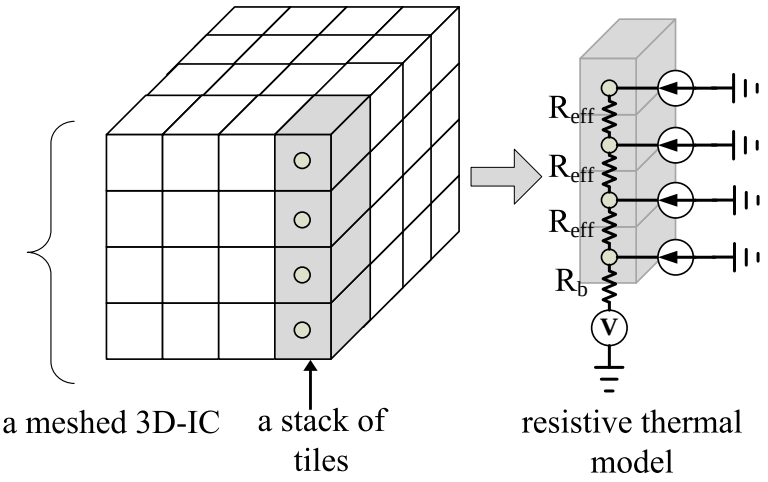


Figure 3.5: Thermal model.

to the following practical assumptions. First, we only consider the functional blocks as the heat sources of a 3D-IC, since most of heat comes from functional blocks. Second, we assume the whole thermal system is in steady state, and the power density of each block is uniform and time-invariant. Third, the top and four sides of the 3D-IC are adiabatic, and the bottom is held isothermally at 25 °C. Base on these assumptions, the resistive thermal network model is adopted to analyze the temperature. The 3D-IC is meshed into stacks of tiles, as shown in Figure 3.5. In this figure, each node represents a tile, and is connected with its vertical neighbors through effective thermal resistors R_{eff}. The heat generated by each tile is formulated as an ideal current source. The bottom tiles are attached to a common heat sink with thermal resistors R_b. In this thermal system, we can compute the temperature of each node by solving linear equations.

3.3 Algorithm

We propose a two-stage approach to solve the problem formulated in Section 3.2.4. The first stage is a simulated annealing based floorplanning algorithm. During this stage, we simultaneously determine the coordinate of each block and plan TSVs for minimizing wirelength under the fixed-outline constraint. The second stage refines the TSV planning result produced in the first stage. The overall flow of our algorithm is shown in Figure 3.6. First, an initial solution is randomly generated. In the first stage, an array of sequence pairs [30] is used to represent a 3D floorplan. Each sequence pair in the array represents a floorplan of the corresponding tier. Then, we perturb the solution by the operations proposed in Section 3.3.1. Then, two different TSV planning strategies are conducted according to the annealing temperature. The cost of a floorplan is given by the following equation:

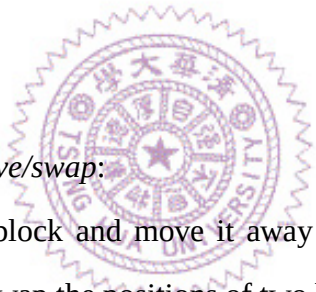
$$cost = area + \alpha \cdot wirelength + \beta \cdot AR\_penalty \tag{3.1}$$

where $area$, $wirelength$, and $AR\_penalty$ refer to the area, the total wirelength, and the penalty of aspect ratio of a floorplan, respectively. Notice that, the $wirelength$ is estimated according to the positions of both TSVs and pins, and the $area$ also includes the area of TSVs. The linear cost function is used to determine whether we accept the current floorplan or not. If the current floorplan is rejected, we restore the floorplan to previous floorplan. Stage one ends when a stopping criterion is met. Based on the floorplan produced by stage one, we formulate the TSV re-assignment problem as a minimum cost maximum flow problem which further minimizes the total wirelength. In the following, we will first give detailed

descriptions of step 1, *perturbation_of_solution*, and step 2, *TSV_planning* of stage one. Then, stage two will be discussed thoroughly.

3.3.1 Perturbation of Solution

To achieve better local search in the solution space, we adopt the slack-based movements proposed by Adya *et al.* [27]. The horizontal and vertical spatial slacks of each block are computed according to constraint graphs in a similar way we compute timing slack in static timing analysis. Then the spatial slacks are used to guide the selection of perturbations. We extend their perturbation operations to 3D floorplanning and design several new perturbation operations to improve successful rate of satisfying the fixed outline constraint. The perturbations are listed as follows:



1. *Random inter-tier move/swap*:

   Randomly choose a block and move it away from the tier where it was placed, or randomly swap the positions of two blocks.

2. *Area balancing move*:

   The imbalance in area utilization among tiers results in a sparse floorplan that has much white space. To facilitate the convergence and balance the area utilization among tiers, we give a higher probability for blocks in a congested tier to move them to a less congested tier.

3. *Slack based inter-tier move/swap*:

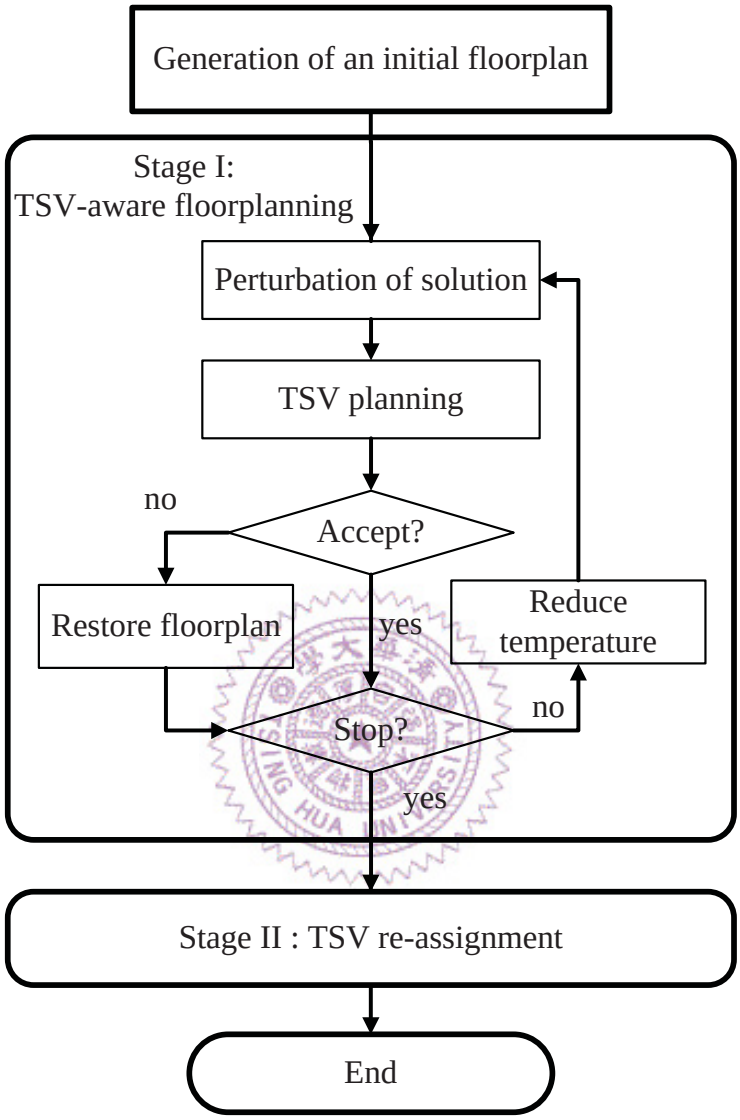


Figure 3.6: Overall flow.

Move a block with negative slack close to another block with positive slack, or swap a block with negative slack with another block with positive slack.

3.3.2 TSV Planning

Based on the floorplan F produced by a perturbation described in the last section, TSVs are then placed into appropriate TSV-blocks. We adopt the concept used in buffer planning [31] and use two algorithm to plan TSVs at different annealing temperature. Initially the annealing temperature is high. The difference between the intermediate and the final floorplan is quite large. Using a precise TSV planning scheme is time-consuming and less meaningful. Rather than a precise TSV planning, we use a simple and efficient algorithm (*prob_TSV_planning*) which probabilistically estimates the distribution of TSVs. As the annealing temperature cools down, the floorplan gradually converges. We use a detailed TSV planning scheme (*detailed_TSV_planning*) to place TSVs such that wirelength can be more accurately estimated. In the following, we will give descriptions of two algorithms, *prob_TSV_planning* and *detailed_TSV_planning*.

Prob_TSV_planning

The flow chart of *prob_TSV_planning* is shown in Figure 3.7. In this procedure, we do not assign TSVs to particular TSV-blocks precisely. Instead, we adopt the idea of probabilistic analysis [31–33] and compute the probability of a TSV-block that a TSV will be assigned to.

In the first step, we apply a line-sweeping-based algorithm to find the white space of F. All white space blocks are TSV-blocks. Next, we compute the capacity used by a TSV by assigning a TSV to TSV-blocks in a probabilistic way. For a net e_i crossing the j-th tier, its bounding box of pins at the j-th tier may cover several TSV-blocks. Let $candidate(\tau_{i,j}) = \{\omega_{j,k} | \omega_{j,k}$ is a TSV-block whose center falls in the bounding box of $e_i\}$. The TSV, $\tau_{i,j}$, of e_i in tier j can be assigned to any of these TSV-blocks, $candidate(\tau_{i,j})$, without increasing the total wirelength. We assume that the probabilities of $\tau_{i,j}$ assigned to a particular $\omega_{j,k}$ are the same. Then, for a TSV-block, $\omega_{j,k}$, of net e_i, its used area is given by the following equation:

$$used\_area = \frac{a_{TSV}}{|candidte(\tau_{i,j})|} \tag{3.2}$$

We repeat the assignment procedure for all TSVs. In each iteration, the used capacity will be accumulated for a TSV-block. After the procedure of assigning all TSVs to TSV-blocks is completed, we check for any TSV-blocks with excessive assigned capacity. If there are, we expand their widths to accommodate all assigned TSVs.

Since some TSV-blocks are expanded, some blocks may overlap. Thus, in the last step, we adjust the floorplan to remove the overlaps.

Detailed_TSV_planning

The flow chart of *detailed_TSV_planning* is shown in Figure 3.8. First, we decide the TSV-blocks by utilizing white space the same way as we do for *prob_TSV_planning*. Next, for each $\tau_{i,j}$, we find its $candidate(\tau_{i,j})$ and compute the total area of

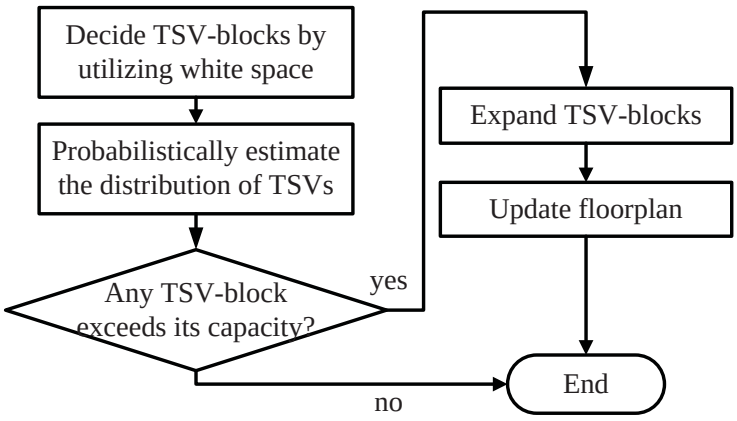


Figure 3.7: Flow chart of prob_TSV_planning.

$candidate(\tau_{i,j})$. We sort TSVs, $\tau_{i,j}$s, by their total area of $candidate(\tau_{i,j})$. The smaller area, the higher the priority of the TSV. The heuristic is based on the observation that the tighter an area of TSV-blocks, the more diffcult it is to place a TSV.

In the third step, we will place a TSV one by one based on its priority. For a TSV, $\tau_{i,j}$, we select a TSV-block from its $candidate(\tau_{i,j})$ to place it. The TSV-block in $candidate(\tau_{i,j})$ that has the largest free capacity will be selected first. If we can place a TSV in any of its $candidate(\tau_{i,j})$ TSV-blocks, the wirelength of e_i will not increase. This is because all the TSV-blocks in $candidate(\tau_{i,j})$ are in the bounding box of e_i's pins. However, if all TSV-blocks in $candidate(\tau_{i,j})$ are full, we are not allowed to assign $\tau_{i,j}$ without increasing the wirelength of e_i or the size of a TSV-block. In this case, we will delay the assignment of $\tau_{i,j}$ and proceed to the next TSV. The delayed TSV will be processed in the *fine_tune_TSV_planning* step. We continue to assign all TSVs until all TSVs are processed.

In the fourth step, *fine_tune_TSV_planning*, we are to place those difficult-to-place TSVs. For a TSV,$\tau_{i,j}$, we can either increase the size of a TSV-block in $candidate(\tau_{i,j})$ or the wirelength by planning TSV outside the bounding box. We will choose the one with the least cost (the cost function will be defined in the following paragraph).

Consider the assignment of a TSV τ where all TSV-blocks in $candidate(\tau)$ are all fully occupied, as shown in Figure 3.9(a). Let e be the net of τ and the rectangle R be the bounding box of the pins of e. a is a candidate TSV-block in R, and b is another TSV-block not in R. Without detouring net e, one may expand the width of a by w_{TSV} and then assign τ to a as shown in Figure 3.9(b). Suppose there are three nets crossing through a. Expanding a will increase the wirelength of these three nets by w_{TSV}. In total, the wirelength is increased by $3w_{TSV}$. On the contrary, assigning this TSV to b will detour net e by $2w_{TSV}$ (assume $h_{TSV} = w_{TSV}$). Thus, in the *fine_tune_TSV_planning* step, the cost of a TSV-block is given by the following equation:

$$cost(\tau_{i,j}, \omega_{j,k}) = B \cdot w_{TSV} \cdot congestion(\omega_{j,k}) + detour(\tau_{i,j}, \omega_{j,k}) \quad (3.3)$$

where $\tau_{i,j}$ is the TSV that we are going to place and $\omega_{j,k}$ is the TSV-block $\tau_{i,j}$ will be placed into. In this equation, B is 1 or 0, where 1 means $\omega_{j,k}$ is expanded and 0 otherwise. $congestion(\omega_{j,k})$ is the number of wires crossing $\omega_{j,k}$, and $detour(\tau_{i,j}, \omega_{j,k})$ is the detouring distance of e_i. Note that if $\omega_{j,k}$ is not expanded, then the first term is zero. We compute $cost(\tau_{i,j}, \omega_{j,k})$'s for all $\omega_{j,k}$'s and

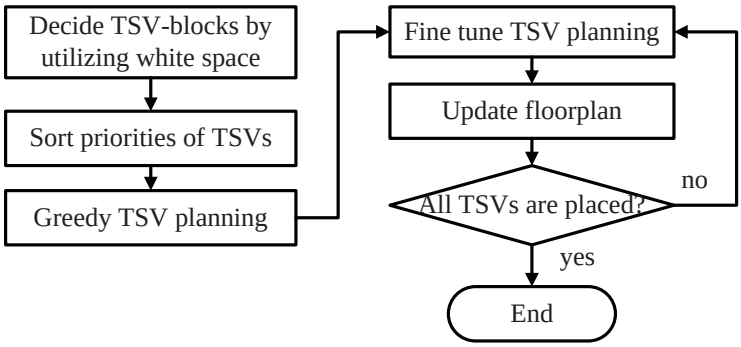


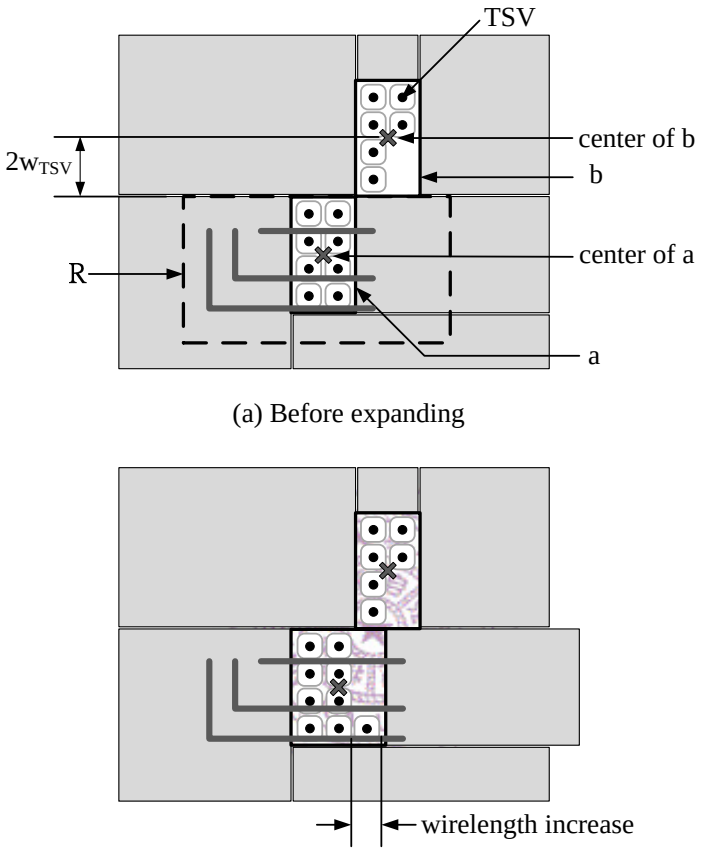
Figure 3.8: Flow chart of detailed_TSV_planning.

select the one with the least cost. If a TSV-block is expanded, the floorplan is updated accordingly. The whole procedure ends when all TSVs are assigned.

3.3.3 TSV Re-Assignment

In the first stage, TSVs are placed one-by-one during floorplanning. The positions of TSVs are not globally optimal because TSVs are not considered simultaneously. For this reason, we will propose our stage two algorithm to improve the result of TSV planning. Note that the wirelength is included in the cost function (Equation 3.1) used to select floorplan solution. The inaccurate estimation of wirelength may degrade the optimality of the floorplan result.

Without altering the floorplan, stage two refines the TSV planning to further reduce the total wirelength. Instead of replacing TSVs individually, stage two simultaneously reassigns TSVs to all available TSV-blocks. Since positions of all modules and TSV-blocks are all fixed in this stage, we can model the TSV re-assignment as a minimum cost maximum flow problem.



(a) Before expanding

(b) After expanding

Figure 3.9: Expansion and detouring.

First, we describe how to build the flow network. The flow network can be represented by a directed graph $G(V, E)$, where node set $V = \{s \cup t \cup D \cup C\}$ and edge set $E = \{IE \cup AE \cup OE\}$. s and t denote the source node and sink node. One node $d_{i,j} \in D$ corresponds to one TSV $\tau_{i,j}$, and one node $c_{i,k} \in C$ corresponds to one TSV-block $\omega_{i,k}$. For clarity, we categorize edges into three groups: *incoming edges* IE, *assignment edges* AE and *outgoing edges* OE. *Incoming edges* are constructed from source s to nodes in D. *Assignment edges* are constructed from D to C to depict all possible TSV assignment configurations. Therefore, an edge from $d_{i,j}$ to $c_{i,k}$ is built if $\tau_{i,j}$ can be assigned to $\omega_{i,k}$. *Outgoing edges* are constructed from nodes in C to sink node t. Figure 3.10(a) is an illustration of the flow network containing 4 TSVs and 3 TSV-blocks. The costs and capacities of all edges are set as follows. For each *incoming edge*, the cost and capacity are set to be 0 and 1, respectively. For each *assignment edge* $(d_{i,j}, c_{i,k})$, the cost is the amount of detour wirelength when TSV $\tau_{i,j}$ is placed in TSV-block $\omega_{i,k}$, and its capacity is 1. Finally, the costs of all *outgoing edges* are 0, and the capacity of *outgoing edge* $(c_{i,k}, t)$ is set as the capacity of TSV-block $\omega_{i,k}$.

Take the example in Figure 3.10(a) as a demonstration for computing the costs of *assignment edges*. Let the physical locations of TSV-blocks, $\omega_{2,1}$ and $\omega_{2,2}$, and pin bounding boxes be shown in Figure 3.10(b), and TSVs $\tau_{2,1}$ and $\tau_{2,2}$ in tier two and their connecting pins in tier one and tier two. The dashed rectangles are the pin bounding boxes of two nets. Circles and rhombuses indicate pins in tier 1 and tier 2, respectively. The solid rectangles are two TSV-blocks. If the TSV $\tau_{2,2}$ is placed in TSV-block $\omega_{2,1}$, the wirelength of $\tau_{2,2}$'s net is unchanged. On the

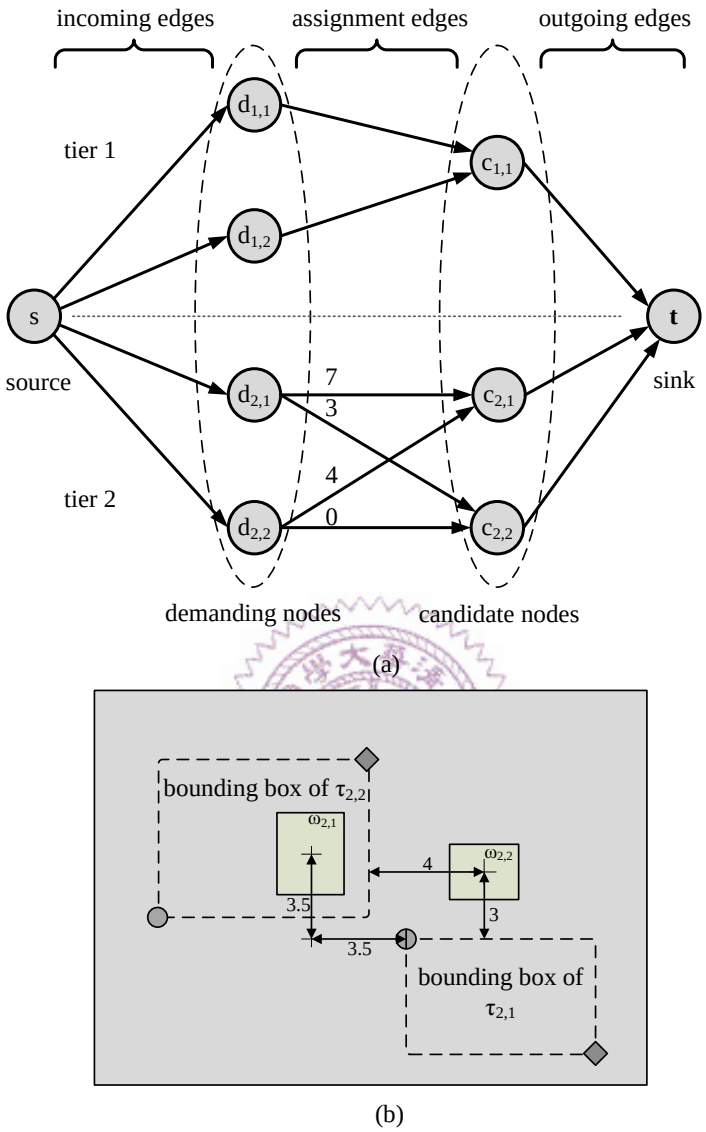


Figure 3.10: A flow network and the corresponding placement.

contrary, placing $\tau_{2,2}$ in $\omega_{2,2}$ will widen its net bounding box by 4 units. Thus, the costs of edges $(d_{2,2}, c_{2,1})$ and $(d_{2,2}, c_{2,2})$ are 0 and 4, respectively. Similarly, the cost of edge $(d_{2,1}, c_{2,2})$ is 3. Note that assigning $\tau_{2,1}$ to $\omega_{2,1}$ not only increases the width of bounding box by 3.5 units, but also increases the height by 3.5 units. Therefore, the cost of edge $(d_{2,1}, c_{2,1})$ is 7. The costs of the *assignment edges* in tier 2 are labeled above those edges in Figure 3.10(a).

The graph has a maximum flow only when each flow of *incoming edge* is 1. Since the entering flow is equal to the leaving flow at each node $d_{i,j}$ in D, there is one and only one of $d_{i,j}$'s *assignment edges* with flow of 1. It implies one TSV will be assigned to one TSV-block. In addition, the entering flow of each candidate node $c_{i,k}$ cannot exceed the capacity of the *outgoing edge* attached to $c_{i,k}$. Therefore, the number of TSVs placed in the corresponding TSV-block will also not exceed its capacity. The minimum cost maximum flow problem can be optimally solved in polynomial time. In our implementation, this problem is solved with LEDA package [34]. The TSV planning result can then be obtained by inspecting the *assignment edges* with a flow of 1. Since the TSV planning solution of stage one is in the solution space of our TSV replanning problem , the new solution is at least as good as the original solution in terms of wirelength.

3.4 Experiment Results

The proposed algorithm is implemented in C++ based on Parquet [35] and run on an Intel 3.0GHz machine with Linux. The MCNC and GSRC benchmarks with 3 and 4 tiers of chip stacks were used to test our algorithm. The thickness of each

tier was set to 20μm [14]. The aspect ratio was 1. Normally, 15% [27] [28] of white space is used for 2D fixed-outline floorplanning problem. However, for 3D floorplanning problems, each tier contains much fewer blocks, and the number of geometric combinations of blocks also decreases significantly. Thus, we use more white space for multi-tier floorplans. Additionally, the more tiers a chip contains, the more TSVs are required. Therefore, we used 15% and 20% white space for 3-tier and 4-tier floorplans, respectively. In MCNC benchmarks, the size of TSV was set to 20μm by 20μm. Since the GSRC benchmarks are artificial circuits with small blocks and more nets, we shrunk down the size of TSV to 3μm by 3μm such that the ratios of TSV size to block size are compatible in both GSRC benchmarks and MCNC benchmarks. In our thermal analysis, the power density of each block was randomly assigned from $10^5 W/m^2$ to $10^7 W/m^2$, and the thermal resistance per square meter was set to $1.57 \times 10^{-6} K/W$.

To demonstrate the effectiveness of our two-stage algorithm, we will show the experimental results of both stages in the following two subsections, individually. All the reported data in our experiments are collected from 10 independent runs.

3.4.1 Results of TSV-Aware Floorplanning

First, we compared our algorithm (without TSV-reassignment) against another TSV-aware algorithm proposed by He *et al.* [15]. Table 3.2 shows the results. The two major columns labeled *He's* and *ours* show the He's results and ours, respectively. The *M65*, *M99*, *M147* and *M198* are artificial test cases created by He *et al*. In He's work [15], the authors magnified both the width and height

of *ami33*, *M65*, *M99* and *M198* by 30 times, and *ami49* and *M147* by 5 times. To make a fair comparison, we scaled down He's floorplans to the original size. The columns labeled *WL* are the total wirelengths estimated using HPWL. The columns labeled *area* are the total areas of the floorplans. The column labeled *inBox* indicates the percentages of nets whose signal TSVs are totally placed into the white space within the bounding box of net pins. Finally, the columns labeled *runtime* show the runtimes. As we can see from He's results, their TSV-aware algorithm cannot guarantee 100% of the TSVs being put in the bounding boxes of nets. In the case of *ami33*, 26% of TSVs cannot be put in the bounding boxes. Furthermore, our algorithm outperforms He's algorithm by 68% and 9% in terms of wirelength and area, respectively.

To analyze the benefit of TSV planning during floorplanning, we compared algorithms with and without the consideration of TSV planning (i.e. *TSV-aware* and *TSV-unaware*). The *TSV-unaware* algorithm minimizes wirelength under a fixed-outline constraint. It uses a half-perimeter bounding box model to estimate wirelength during the floorplan stage without considering the TSV planning problem. After the floorplanning is done, we apply the $detailed\_TSV\_planning$ algorithm to plan TSVs. The reported wirelengths are computed by the model proposed in Section 3.2.2 after the post-processing TSV planning.

Table 3.3 shows the results of successful rates and total wirelengths. The first two columns give the numbers of tiers and the names of benchmark circuits. The column labeled *S.Rate* shows the successful rates. Here, a floorplan is said to be successful only if it meets the fixed-outline constraint. The average wirelengths

are reported in the column labeled *avg WL*. For the *TSV-unaware* algorithm, the percentages of nets with signal TSVs completely placed within the bounding box of net pins are also shown in the column labeled *inBox*. In addition, the wirelength differences between the bounding boxes of nets with and without TSVs are shown in the column labeled *error*. Here *error* is computed as the ratio of wirelength differences to the wirelength considering signal TSVs. Finally, the numbers of TSVs and runtimes are shown in the columns labeled *TSV* and *runtime*, respectively. Since the test cases *apte*, *xerox*, and *hp* in MCNC benchmark contain only a small number of blocks, they are not good examples for 3D-IC, and are excluded in our benchmark set. The experimental results show that our stage one (*TSV-aware*) algorithm outperforms the *TSV-unaware* algorithm in successful rate by 57%. Note that our stage one algorithm and *TSV-unaware* use the same algorithm to plan TSVs. The only difference between these two algorithms is that the former plans TSV in the flooplanning stage, while the latter plans TSV after the floorplanning stage. In addition, all the floorplans produced by *TSV-unaware* algorithm meet the fixed-outline constraint before planning TSVs. During the post-process TSV planning, the successful rate is reduced to 42%, the reason being given as follows. Since the *TSV-unaware* algorithm does not consider the TSV planning during floorplanning, the distribution of white space is inconsistent with the requirement of TSVs. Up to 49.9% of signal TSVs cannot be placed within the bounding boxes. Therefore, many white spaces need to be expanded to accommodate those TSVs during the post-processing TSV planning. As a result, the floorplans are likely to violate the fixed-outline constraint. Compared to the

Table 3.2: Comparison between algorithm proposed in [15] and our algorithm.

		He's				ours		
#tier	circuit	WL	area(mm^2)	inBox	runtime(s)	WL	area(mm^2)	runtime(s)
4	ami33	63773	1.44	73.71 %	75.97	45179	1.37	42.46
4	ami49	1127330	45.10	87.54 %	144.88	585804	41.71	184.63
4	M65	129394	2.89	98.28 %	192.53	79857	2.64	209.38
4	M99	181608	4.48	100.00 %	380.11	108528	3.95	544.93
4	M147	3557120	132.19	100.00 %	764.34	1718420	121.09	2633.63
4	M198	314953	8.66	100.00 %	1417.50	180042	7.92	2945.89
	avg	1.68	1.09	93.26 %	0.83	1	1.00	1.00

TSV-unaware algorithm, the average wirelength and the number of TSVs of our result are reduced by 22.3% and 15.0%, respectively.

To prove that our signal TSV-aware floorplanner can readily cope with the thermal issue, we modified our algorithm to consider temperature by integrating the thermal analysis step described in Section 3.2.5 and compare the results of the modified algorithm with that of the floorplanner without thermal consideration. The experimental results are shown in Table 3.4. The column labeled *thermal-aware* and *thermal-unaware* show the results with and without thermal consideration, respectively. Compared to the thermal-unaware algorithm, our thermal-aware algorithm effectively reduces the peak temperature by 37.0% at the expense of a 3.9% increase in wirelength. The promising result guarantees that our algorithm is extendable to thermal optimization as well.

Table 3.3: Comparison between TSV-aware and TSV-unware algorithms.

		TSV-aware				TSV-unaware					
#tier	circuit	S.Rate	avg WL	TSV	runtime(s)	S.Rate	avg WL	inBox	error	TSV	runtime(s)
3	ami33	100%	47731	97.2	41.17	30%	60438	58.0%	31.0%	132.5	1.82
3	ami49	100%	690825	361.2	172.92	90%	890588	50.1%	30.2%	443.5	5.51
3	n100	100%	160825	833.2	1195.51	80%	157480	73.5%	14.8%	888.8	22.68
3	n200	100%	310924	1509.1	7720.45	0%	339768	66.9%	26.9%	1689.5	87.38
3	n300	100%	424585	1899.7	21155.10	0%	440954	83.7%	16.5%	2019.3	159.02
4	ami33	100%	45179	141.0	42.46	50%	60772	56.6%	39.2%	174.7	2.00
4	ami49	90%	585804	435.6	184.63	70%	773076	59.0%	21.4%	505.4	4.80
4	n100	100%	148748	1171.4	1306.39	90%	165940	69.9%	18.7%	1290.5	23.38
4	n200	100%	291091	2179.0	8237.10	0%	367602	60.5%	34.1%	2431.5	94.45
4	n300	100%	391694	2730.6	21450.50	10%	448905	80.9%	19.0%	2865.0	234.12
	avg	0.99	1	1		0.42	1.223	0.659	0.252	1.15	

Table 3.4: Comparison between algorithm with and without thermal consideration.

		thermal-unaware			*thermal-aware*		
#tier	circuit	S.Rate	avg WL	maxT	S.Rate	avg WL	maxT
3	ami33	100%	47731	76.0	100%	51859	59.4
3	ami49	100%	690825	59.5	100%	694622	48.1
3	n100	100%	160825	85.3	100%	164975	47.8
3	n200	100%	310924	93.6	100%	314450	68.1
3	n300	100%	424585	90.3	100%	435885	50.5
4	ami33	100%	45179	85.2	100%	49934	67.4
4	ami49	90%	585804	76.6	80%	610077	60.2
4	n100	100%	148748	130.3	100%	155780	49.8
4	n200	100%	291091	114.6	100%	298025	58.7
4	n300	100%	391694	132.1	100%	396262	52.0
	avg	0.98	1	1	0.95	1.039	0.630

Table 3.5: The total wirelength improvement rate of TSV re-assignment.

#tier	circuit	avg imprv.	min imprv.	max imprv.	runtime(s)
3	ami33	4.33%	1.31%	8.61%	< 1
3	ami49	0.26%	0.01%	1.02%	< 1
3	n100	1.67%	0.78%	2.86%	< 1
3	n200	4.54%	2.80%	7.13%	< 1
3	n300	2.40%	1.41%	4.94%	< 1
4	ami33	2.93%	0.94%	5.68%	< 1
4	ami49	1.35%	0.00%	3.56%	< 1
4	n100	5.16%	1.63%	7.16%	< 1
4	n200	6.68%	2.71%	11.39%	< 1
4	n300	5.17%	3.61%	8.45%	< 1
	avg	3.45%	—	—	

3.4.2 Results of TSV Re-Assignment

To understand how effective our TSV re-assignment is, we compared the experimental results of stage two with those of stage one. The experimental results are presented in Table 3.5. The columns, in order, give numbers of tiers, the names of benchmarks, average improvement rates, minimum improvement rates and maximum improvement rates. Note that stage two only re-assigns TSVs to existing TSV-blocks, and does not change the floorplan results at all. As is shown, stage two further reduces the wirelength by 3.45% without increasing area.

3.4.3 Sensitivity Analysis on Size of TSVs

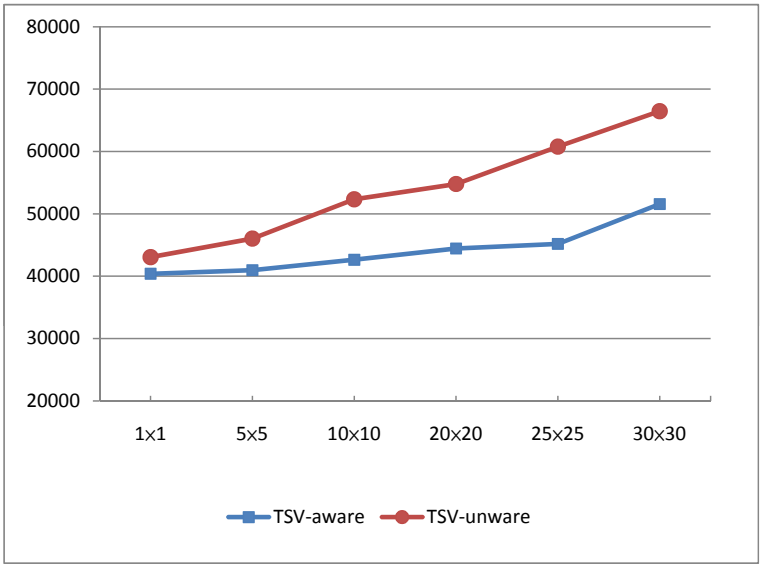
Since the size of TSV depends on different 3D technologies, we also explore how the size of TSV affects wirelengths of floorplans. We performed our TSV-aware floorplaning algorithm (i.e., stage one algorithm) with different sizes of

TSVs for 4-tier cases, and compared the results with those of the TSV-unaware algorithm used in Section 3.4.1. Note that the TSV planning algorithms used in both the TSV-aware and TSV-unaware algorithms are exactly the same. The only difference between these two floorplaning algorithms is that the TSV-aware one simultaneously plans blocks as well as TSVs while the TSV-unaware one plans TSVs after the floorplaning stage.

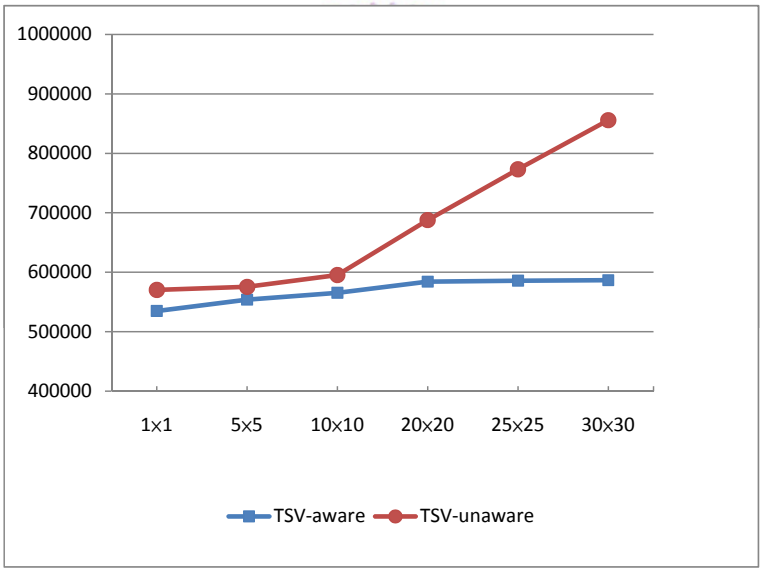
In this experiment, we set the size of TSV from $1\mu m$ to $30\mu m$. For all the cases, the white space ratios are set to 20%, except the case of *ami33* with TSV size of $30\mu m \times 30\mu m$. In this exceptional case, TSVs consume more than 10% of the block area and the large TSVs cannot be put into fragmental white space. Therefore, we set the white space ratio to 25%. Figure 3.11 shows the experimental results. Each node represents an average wirelength of 10 independent runs. As we can see from these curves, when the TSV size is small, the difference between the TSV-aware and TSV-unaware algorithm is insignificant. However, as the size grows, the wirelength of the TSV-unaware algorithm increases rapidly. On the other hand, the wirelength of TSV-aware algorithm increases less significantly. Thus, the wirelength difference increases with the increasing size of TSVs. Obviously, planning TSVs in floorplaning stage is important when large TSVs are used.

3.5 Summary

In this chapter, we have studied the signal TSV planning for 3D-IC. Conventionally, the placement of modules is first determined in a floorplan stage. Then, TSVs



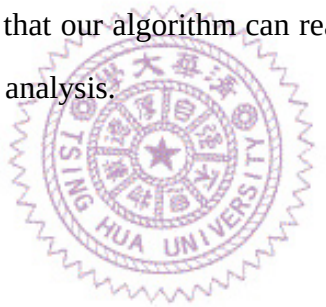
(a) ami33



(b) ami49

Figure 3.11: Average wirelengths with different TSV sizes.

are deployed within the white space. Lacking the information of TSVs, a floorplanner estimates the wirelength of a net by the half-perimeter of the bounding box of the net pins only. However, the white space distribution may be inconsistent with the requirement of TSV deployment, which in turn increases the wirelength. To cope with this problem, we have proposed a 3D fixed-outline floorplanning algorithm to simultaneously place modules and plan TSVs for wirelength reduction. Experimental results show that our algorithm achieves a high successful rate. Compared to the average wirelength of a post-processing TSV placement algorithm, our result is shorter by 22.3%. In addition, we have also proposed a TSV re-assignment algorithm to refine a TSV planning result. Experimental results show that the wirelength can be reduced up to 11.39% without area overhead. Furthermore, we also show that our algorithm can readily cope with the thermal issue by integrating thermal analysis.



Chapter 4

An Evaluation of Trade-off among Wirelength, Number of Through-Silicon Via and Placement in 3D-ICs

Recently, 3D-IC has drawn much attention and been studied from many perspectives including fabrication processes [36, 37], production cost [38], yield rate [39], and thermal issues [17–19]. Prior research found that interconnects are significantly reduced with the existence of vertical connections. However, in the rudimentary analysis, the authors [21, 22, 40–42] viewed the vertical connections as volumeless objects and did not consider achievable size of TSV in present technology. Therefore, the result is over-optimistic. Although TSV is able to provide the highest density of inter-chip connections, area occupied by TSVs still cannot be ignored. In reality, applying too many TSVs significantly increases the chip size and extends the lengths of interconnects. Therefore, using more TSVs does not necessarily favor the performance of a design. On the contrary, min-

imizing the number of TSVs diminishes the benefit brought by TSV. Recently, there is plenty of literature discussing how wirelengths varies with the number of TSVs. Most of their conclusions are drawn without considering the area of TSVs. Among them, only D. H. Kim *et al.* [43, 44] have studied the impact of TSV area on 3D-IC. However, they only consider single TSV size. In fact, the impact of TSV on a design depends on the technology being used. Different technologies often have different TSV sizes. For example, the size of TSV can range from $10\mu m$ by $10\mu m$ [14] in a via-first technology to $200\mu m$ by $200\mu m$ [37] in a via-last technology. Therefore, it is crucial to explore how different TSV technologies affect designs of 3D-ICs. Additionally, D. H. Kim *et al.* [43] only analyzed the relationship between wirelength and number of TSVs focusing on wirelengths of minimum spanning trees (MSTs) and routing trees with least number TSVs. To make a comprehensive study, it is also important to study the trade-off between these two extreme routing topologies.

In this chapter we will study the trade-off among wirelength, number of TSVs, and size of TSV. Moreover, we propose a parameterized partition algorithm to study the trade-off between different placement and the number of TSVs.

The rest of the chapter is organized as follows. Section 4.1 presents the motivation of this work. Section 4.2 describes flow of our analysis and the detail of our evaluation algorithm. The experimental results are shown and discussed in Section 4.3. Finally, Section 4.4 summarizes this chapter.

4.1 Motivation

Previous research [21, 22, 40–42] has shown the benefit of transforming designs from 2D to 3D and examined the tradeoffs between wirelengths and number of TSVs in several 3D placement approaches. Without considering the area overhead of TSVs, the wirelength of 3D IC reduces substantially with the increase in the number TSVs. In an experiment [22], more than 6,000 TSVs were used in a design with $0.06mm^2$ of core area. Suppose that the size of the applied TSV is $10\mu m \times 10\mu m$. The TSVs will occupy an area as large as the core. To accommodate these TSVs, the die size must expand twice. Consequently, the cells are forced to spread more sparsely and the interconnects are also extended significantly. Therefore, using excess TSVs will pose a negative effect to a 3D design.

To investigate the impact of TSVs, we performed an analysis based on the experimental result conducted by J. Cong *et al.* [22]. The area of each die is computed as the sum of both the cell area and the TSV area, and the wirelengths is scaled with the die size. Table 4.1 shows the usage of TSVs. The row labeled *strategy* indicates different 3D placement methods used in [22]. The average numbers of TSVs are normalized and presented in the row labeled *TSV usage*. Figure 4.1 shows how the average wirelengths vary with different size of TSVs. To make a fair comparison, all the wirelengths of 3D placements are normalized with that of 2D placements. Each point in the figure represents an average of 10 placement results. The curve labeled *original* indicates the normalized average wirelengths reported in the previous work. Without considering the area overhead of TSVs,

Table 4.1: Benchmarks

strategy	2D	Folding-2	Folding-4	LST(8x8)	LST(10%)
TSV usage	NA	1.00	1.79	4.8	13.92

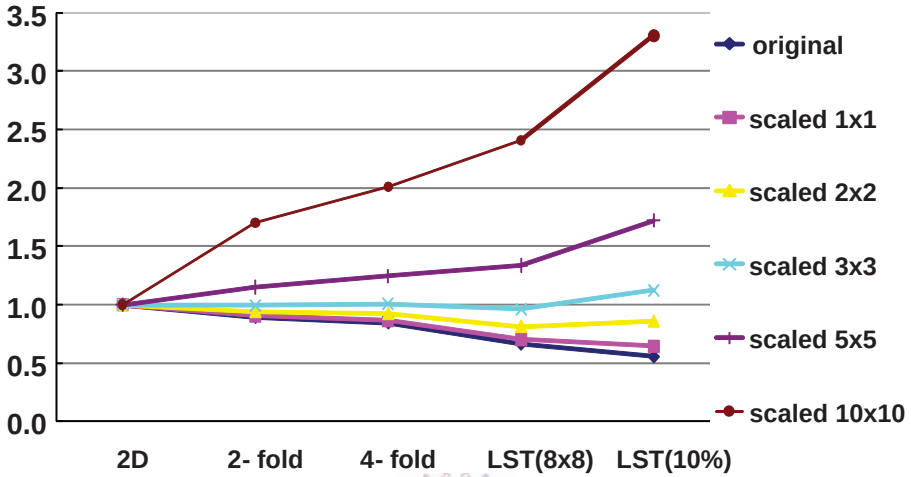


Figure 4.1: The impact of TSV on wirelength of IBM-PLACE2 benchmarks [45].

the average wirelength decreases with increase in the number of TSVs used in placements [22]. Up to 44% of wirelength reduction is obtained in the result of *LST(10%)* [22]. However, with the size of TSV growing shown in Figure 4.1, the wirelength gradually increases due to the increase in die size. When the size of TSV is $3\mu m \times 3\mu m$, the benefit of TSV on wirelength reduction is virtually compensated by the area overhead. Beyond this size, the wirelengths are even longer than that of 2D placements.

As shown in the aforementioned analysis, the wirelength of a design varies with the number of TSVs as well as the size of TSV. To fully utilize the advantage

of TSV technology, we should explore the 3D design space with different TSV parameters. Therefore, in this chapter, we will investigate the trade-offs among wirelength, number of TSVs and size of TSV.

4.2 Evaluation of TSV and Wirelength

As discussed in the previous section, there is a trade-off between wirelength and number of TSVs. Therefore, it is important to understand how much benefit we can obtain over a wide range of TSV numbers. Since different net topologies always lead to different wirelengths and different numbers of TSVs, it is our interest to analyze the wirelength of 3-D net with different net topology. Therefore, in this chapter, we will describe three wirelength analyses with different net topologies and illustrate our evaluation algorithms individually. For clarity, the notation used in the following section is given in Table 4.2.

Table 4.2: Notation list

k	the number of tiers
n_i	the i-th net
gc_i	the complete graph of n_i
$e_{i,j}$	the j-th edge in gc_i
$Ne_{i,j}$	the number of TSVs of $e_{i,j}$
$N_{MST}(n_i)$	the number of TSVs in minimum spanning tree of n_i
$N_{LTT}(n_i)$	the number of TSVs in the least-TSV tree of n_i

```
Nmax = 0;
Nmin = 0;
total_wirelength = 0;
total_TSV = 0;
for (i = 1 to number of nets) {
   Nmax = Nmax + N_MST(n_i);
   Nmin = Nmin + N_LTT(n_i);
   primorial_tree_i = MST of n_i;
   derived_tree_i = replace_edge(primorial_tree_i);
   increment_i = length(derived_tree_i) − length(primordial_tree_i);
                 // length(T) returns ∞ if T does not exist.
   total_wirelength = total_wirelength + length(primorial_tree_i);
   total_TSV = total_TSV + number_of_TSV(primorial_tree_i);
}

for(p = N_max to N_min) {
   min_increment = ∞;
   for(j = 1 to number of nets)
      if(min_increment > increment_j)
         min_increment = increment_j;
         k = j;
   }
      total_wirelength = total_wirelength + increment_k;
      total_TSV = total_TSV − 1;
      primorial_tree_k = derived_tree_k;
      derived_tree_k = replace_edge(primorial_tree_k);
      increment_k = length(derived_tree_k) − length(routing_tree_k);
}
```

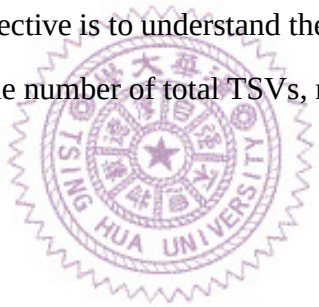
Figure 4.2: Pseudo code of spanning tree construction

4.2.1 Wirelength of 3-D Spanning Tree

Compared to a 2D-IC, TSV provides one more dimension in the z-axis for routing. Therefore, the routing topology is more flexible and the wirelength of a 3-D net can be potentially reduced when more TSVs are used. To investigate how the wirelengths of 3-D spanning trees change with number of TSVs in a fixed 3-D placement, we formulate the problem of our first experiment as follows.

Experiment I:

Given (1) a set of net pins and their positions; (2) a netlist (3) the size of TSV; (4) the die size of the 3D-IC; (5) the low bound, N_{min}, and the upper bound, N_{max}, of number of TSVs, our objective is to understand the relationship between minimum total wirelength and the number of total TSVs, ranging from N_{min} to N_{max}.



Evaluation Algorithm I:

To solve problem I, the first thing is to find N_{min} and N_{max}. Assume we have a 3-D net spanning from tier a to b, where $a < b$. Since we need at least one TSV to link the interconnect in adjacent tiers, the 3-D net requires at least $b - a$ TSVs. Therefore, N_{min} is the sum of least TSV numbers of all nets. For convenience, we refer the net topology with least TSVs as *LTT* (least-TSV topology). Since our interest is to analyze the wirelengths of spanning trees, we set N_{max} to the sum of TSV numbers required by MSTs of all nets.

Since a real design often contains considerable number of nets, finding a set of spanning trees with least total wirelength for every single number of TSVs is time-consuming. Therefore, we propose a divide-and-conquer heuristic to solve problem I. Before we present out heuristic algorithm, we define a *derived tree* DT of spanning tree T as a spanning tree which has only one edge different with T. Tree T is called *primordial tree* of the derived tree DT.

We begin with investigating the wirelength of a single net n_i. We construct the complete graph gc_i of n_i's net pins, assign the weight of each edge as the length between the two terminals of the edge, and then find the MST of n_i using Prim's algorithm. Since MST of n_i is the routing tree of n_i with the shortest wirelength but the maximum number of TSVs, we use it as a starting primordial tree to find derived trees with less number of TSVs.

By the definition of MST, if we can find an edge in gc_i to replace an edge in the primordial MST, the replacement edge must be no shorter than the edge being replaced. To find a derived tree with short wirelength, we first decompose a primordial tree into two subtrees by removing the shortest edge $e_{i,j}$ that contains TSVs. Then we find the shortest replacement edge $e_{i,k}$ that connects the two subtrees and possesses $Ne_{i,j}$ - 1 TSVs. In this way, we can obtain a derived tree with $n-1$ TSVs by modifying a primordial tree with n TSVs. Now given the derived tree with $n-1$ TSVs as a new primordial tree, the same edge replacement can be applied to find a derived tree with $n-2$ TSVs. We repeat the same edge replacement procedure starting with a primordial tree with $N_{MST}(n_i)$ TSVs to $N_{LTT}(n_i)+1$ TSVs. We can obtain $N_{MST}(n_i) - N_{LTT}(n_i)$ derived trees.

Our objective is to observe the minimum total wirelengths for all nets within bounded TSV numbers. Initially, MST of each net is computed, and the total wirelength is the sum of wirelength of all MSTs. Then, we compute the wirelength increment of each net n_i using the edge replacement algorithm described above. To compute the minimum wirelength after removing one TSV, we select the net n_j which induces the least amount of wirelength increment. Figure 4.2 shows the pseudo code to construct spanning trees with TSV numbers ranging from $N_{MST}(n_i)$ to $N_{LTT}(n_i)$.

Our wirelength evaluation is accurate only when TSVs are viewed as infinitesimal objects. However, in reality, the area of TSV cannot be ignored. To consider the area of TSVs, we assume TSVs are evenly distributed in a 3D-IC, and inserting TSVs into a 3D-IC will not alter the relative positions of cells. Therefore, we can compute the wirelengths in the $x-y$ plane by directly scaling the wirelengths. Suppose we have a 3D-IC with N_{TSV} TSVs. The area of a TSV is A_{TSV} and the total area of the 3D-IC is A_{3D-IC}. The scaling factor f is given as follows:

$$f = \sqrt{1 + \frac{N_{TSV} A_{TSV}}{A_{3D-IC}}} \tag{4.1}$$

4.2.2 TSV Impact on Longest Paths

Since critical path delay determines the performance of a design, it is crucial to analyze how TSV impacts the timing of a 3D design. For simplicity, we use the length of longest paths to estimate critical delay and analyze how lengths of longest paths change with number of TSVs and size of TSV. Therefore, the prob-

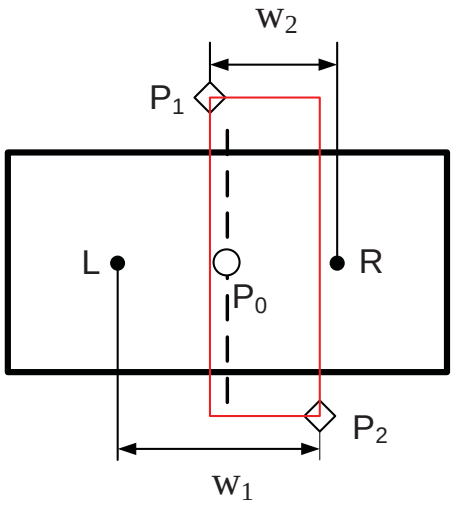


Figure 4.3: The exact net model.

lem of the second experiment is formulated as follows:

Experiment II:

Given (1) a set of net pins and their positions; (2) a set of paths; (3) the size of TSV; (4) the die size of the 3D-IC, our objective is to observe the relationship between lengths of longest paths and the number of total TSVs.

Evaluation Algorithm II:

To solve problem II, we initially route each net with MST. Then, we gradually replace net edges with substitutive edges that have less TSVs and increase least amount of net lengths of longest paths. The edge replacing move continues until no edge has a substitutive edge.

Since a path goes through many nets, directly finding a substitutive edge among all nets of the paths is difficult. Therefore, we adopt a similar divide-and-conquer approach as used in problem I. We first find *derived trees* of each net and obtain the wirelength and the number of TSVs of each *derived tree*. Then, in each edge replacing move, we select the *derived tree* that increases least amount of length of longest paths and find the *derived tree* of the selected *derived tree*. We repeat this procedure and record the lengths of longest paths until no *derived tree* exists.

4.2.3 Wirelength of Different 3-D Placements

Previous section shows how we analyze the wirelengths with different routing topologies under a fixed placement result. Since the wirelength and number of TSVs change with different placements, it is also our interest to exam how placements affect TSV numbers. However, changing the placement induces one more dimension in our analysis space and significantly increases the complexity of our analysis. Therefore, we focus on the *LTT* routing trees in our second analysis. The problem of the third experiment is formulated as follows.

Experiment III:

Given (1) a set of blocks; (2) the size of each block; (3) a netlist (4) the size of TSV, our objective is to observe how the number of total TSVs changes with different placements.

Evaluation Algorithm III:

Since our major interest is to fast investigate the TSV number over a wide range of placements, analyzing 3D-placement results produced by a specific tool is less representative. Therefore, we propose a generic partition-based algorithm to analyze the trend of wirelength variation in a 3D-design.

Our algorithm recursively divides a circuit into two subgroups by utilizing a min-cut objective. Initially, we put all the blocks at the center of the 3D-IC. Then, we recursively divides the circuit into two subgroups by a top-down manner. Different from conventional placers [21, 46] that only minimize the number of cuts between two subgroups, we adopt an exact net model (see Section 4.2.3) and minimize the wirelength. Moreover, since some groups possess nets connecting to the external region, we also use bounding box terminal propagation [47, 48] to guide the bisecting process. The cutting plane can be orthogonal to x, y or z axis, and the detail of choosing cutting plane will be given in Section 4.2.3. After the bisecting, we put the cells to the center of their subgroups accordingly. The bisecting process is continued until group contains one block only. Although the algorithm only produces global placement result, it can generate a series of placements in a short time. Therefore the algorithm is suitable for analyzing the wirelength of 3-D placements.

Exact Net Model

To obtain a better placement result, it is necessary to use an efficient and effective model to estimate the wirelength of nets. Therefore, we use the exact net model

[28] to estimate the wirelength of hyper nets. Figure 4.3 shows the exact net model. Consider a group has a hyper net with three pins p_0, p_1, p_2, where p_1 and p_2 are two points outside the region of the group. The bold rectangle shows the region of the group being partitioned. The dash line shows border of two subgroups. The fine rectangle shows the bounding box of the external pins. The nodes L and R indicate the centers of left and right subgroups, respectively. If p_0 is put in the left subgroup, the wirelength of the hyper net will be w_1; otherwise the wirelength will be w_2. Therefore we create an edge between p_0 and the subgroup center which is closer to the center of the bounding box, and set the cost of the edge as $|w_1 - w_2|$. Before each partitioning, we set the weight of each net by using the exact net model. Then, we find the min-cut partition and divide the original group into two subgroups with minimum wirelength.

Cutting Scenario

The sequence of cutting directions in a placer significantly influences the wirelength of a 3D-IC and the number of TSVs. In some papers [18, 19], the number of TSVs is the objective being minimized; some other papers [40–42] show that the wirelength reduces when more TSVs are used. Therefore, in the following we will first illustrate two extreme cutting scenarios, *z-cut first* and *z-cut* last scenarios, then derive a parameterized cutting scenario used in our algorithm.

The *z-cut first* scenario first divides a circuit into vertical groups, where each group corresponds to a tier in the chip stack. Since the wirelength in z-axis is proportion to the number of TSVs, finding minimum z-cut partitions actually min-

imizes the usage of TSVs. On the other hand, *z-cut last* partition first performs x-cuts and y-cuts, and delays z-cuts to the last. Therefore, the *z-cut last* is actually performing 2-D placement first, then arrange blocks into different tiers vertically. Since the *z-cut last* scenario first performs x and y cuts, it mainly reduces the wirelength in x-y plane and produces more TSVs.

To analyze the wirelength of 3D-IC, we should investigate the placement between the two extreme scenarios. Therefore, we propose a parameterized partition scenario which can gradually change between these two extreme scenarios and produce placements with different number of TSVs. Assume we attempt to place a design with n_{block} cells to k tiers. The circuit will be partitioned recursively by $log_2(n_{block})$ times with $log_2(k)$ z-cuts. Let the first z-cut appear at the b-th position of the bisecting sequence, and the interval between two z-cuts be p. Then, b and p are given in the following equations.

$$b = (log_2(n_{block}) - log_2(k))\alpha + 1 \tag{4.2}$$

$$p = (log_2(n_{block})/log_2(k) - 1) \times (1 - |2\alpha - 1|) \tag{4.3}$$

where α is a real number between 0 and 1. After we decide the first z-cut position and the interval of z-cut, all other positions of the sequence are filled with x-cut and y-cut interleavingly. For example, if $n_{block} = 64$, $k = 4$ and $\alpha = 0$ the sequence of partitioning would be "z-z-x-y-x-y" which is the *z-cut first* partition. If $\alpha = 1$, the sequence would be "x-y-x-y-z-z" which corresponds to the *z-cut*

last. If $\alpha = 0.5$, the sequence would be "x-y-z-x-y-z" which behaves between *z-cut first* and *z-cut last* partitions.

4.3 Experiment Results

The proposed evaluation algorithms are implemented in C++ with *hMetis* [49] for partitioning and run on an Intel 3.0GHz machine with Linux. The IBM-PLACE2 [50] benchmarks with 4 tiers of chip stacks were used to test our algorithm. The thickness of each tier was set to $20\mu m$. All cells in the benchmarks are matched to cells in TSMC $180nm$ standard-cell library (from Artisan Inc.). In the following we will present the results of our evaluations described in the previous section.

4.3.1 Results of Spanning Trees

In this experiment, we aim to demonstrate how total wirelength of 3-D spanning trees varies with TSV size and number of TSV under given placement results. For clarity, we first demonstrate the trade-off of a single case $IBM01$ using a fixed TSV size ($5\mu m$ by $5\mu m$). Figure 4.4 shows the result. To show how much benefit we can obtain from 3-D spanning trees, all wirelengths presented are normalized to the wirelenth using the least number of TSV. The curve *ideal in-plane* shows the ideal wirelength in $x - y$ plane without considering the overhead of TSVs. The curve *factor* shows the scaling factor of wirelength due to the expansion in die size. The evaluated wirelength in x/y plane is computed as the product of *factor* and *ideal in-plane* and is shown by the cure *real in-plane*. Finally, the

3D wirelength indicated by curve *real 3D* is computed as the sum of wirelength in $x - y$ plane and in z axis. As shown in the figure, without considering TSV area, the *ideal in-plane* curve monotonically decreases with the increase in TSV number. Therefore, using MSTs is in favor of reducing wirelength. However, when TSV size is $5\mu m$ by $5\mu m$, the wirelength curve becomes convex and has an optimal point. Before this optimal point, using more TSVs helps the design reducing wirelength. Beyond the optimal point, the penalty of TSV area overhead overtakes the benefit of 3-D spanning trees. Therefore, using excess TSVs only increases wirelength. Next, Figure 4.5 shows how the size of TSV affects the wirelength of *IBM01*. The optimal points are highlighted by the circles. As shown, when the size of TSV grows, the area overhead of TSV gradually increases. Thus, the optimal points shift toward the region with less TSV number. The arrows indicate the movement of optimal points. The same results were observed for other benchmark cases. The existence of optimal point on the wirelength curves indicates that neither *LTT* tree nor MST is the best topology of 3-D routing tree. Next, we want to understand how optimal wirelength changes with the size of TSV. We change the size of TSV from $0\mu m$ by $0\mu m$ to $50\mu m$ by $50\mu m$, and find the optimal wirelength for all test cases. To make a fair comparison, for each case, the optimal wirelength is normalized to the total wirelength of *LTT* spanning trees. The result is presented in Figure 4.6. As shown in the figure, when the size of TSV is smaller than $20\mu m$ by $20\mu m$, the normalized optimal wirelengths are relatively small. When the size of TSV increases, the normalized optimal wirelengths begin to level off and approach to 1. It implies that when

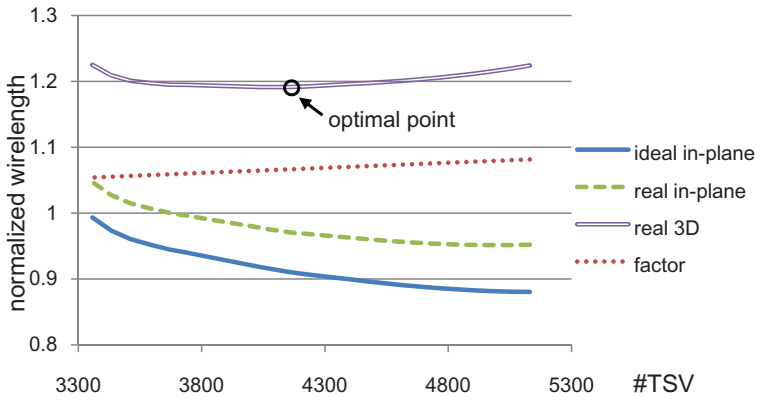


Figure 4.4: The trade-off between wirelength and number of TSV under fixed TSV size in $IBM01$.

TSV size is smaller than $20\mu m$ by $20\mu m$, the best routing topology falls between the *LTT* spanning tree and MST. Beyond this size, we should use *LTT* spanning tree for routing to achieve the minimum wirelength.

4.3.2 Results of TSV Impact on Lengths of Longest Paths

To study how TSV impacts on the lengths of longest paths of given placements, we observe the lengths of longest paths under different total TSV numbers. In this experiment, we choose the top 10% of the longest paths as the longest paths. Since the lengths of longest paths and number of TSVs changes from case to case, we normalized the lengths of longest paths with respect to that of ideal situation in which TSV does not occupy area. Similarly, we also normalized total TSV number to number of TSVs required by the LTT trees.

Figure 4.7(a) plots the average length of longest paths versus normalized number of TSVs under different TSV sizes ($0\mu m \times 0\mu m$, $3\mu m \times 3\mu m$, $5\mu m \times 5\mu m$,

$10\mu m \times 10\mu m$, $15\mu m \times 15\mu m$). Figure 4.7(b) shows the ratio of TSV area to cell area in different TSV numbers. First, we study the path length under ideal situation ($0\mu m \times 0\mu m$). Initially, adding TSVs provides shorter vertical links for longest paths, hence the length of longest paths drops with the increase of TSV number. However, when the normalized TSV number excesses 1.4, virtually all longest paths cannot be further shortened. Adding more TSV only helps the wirelength reduction of non-longest paths. Therefore, the curve starts to level off. When larger TSVs are used, using too many TSVs significantly increases the die size and the length of interconnection. Therefore, a trade-off among the penalty and benefit of TSVs appears. As shown in the figure, the triangles indicate the optimal points of longest path length. Beyond an optimal TSV number, the penalty of TSV area overhead overtakes the benefit of wirelength reduction brought by TSVs. Therefore, path length increases after the optimal point. As shown in the figure, when $10\mu m \times 10\mu m$ TSVs are used, an optimal point appears when normalized TSV number equals to 1.34. When TSV size grows to $15\mu m \times 15\mu m$, the penalty of TSVs is even larger and pushes the optimal normalized TSV number forward to 1.28.

This experimental result implies that we only need to route the longest paths with MSTs to reduce the length of longest paths, and route non-longest paths with least amount of TSVs to reduce the area overhead.

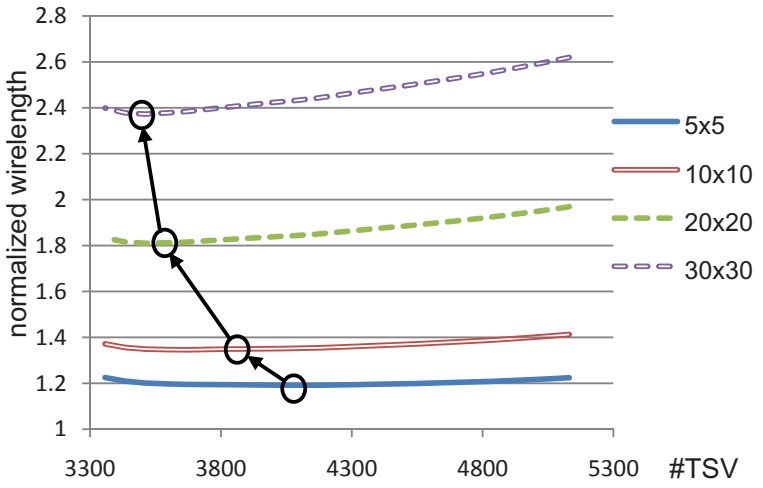


Figure 4.5: The trade-off between wirelength and size of TSV in $IBM01$.

4.3.3 Results of Different Placements

Although previous experiment has shown the trade-offs among wirelength, TSV size, and number of TSVs, the analysis is conducted with given 3-D placements. To make a comprehensive study, it is necessary to understand how placement affects the number of TSVs with a fixed routing tree topology. In the following, we will present the result of the evaluation described in Section 4.2.3 with 4-tier placements. Since we use parameterized cutting scenario to control our placer to produce different placement results, we first show how the number of TSVs changes with the parameter α. Figure 4.8 plots the TSV numbers versus α. Since the number of TSVs changes from test case to test case, we normalized it to the maximum number of TSVs required by the test case. When α changes from 0 to 1.0, our cutting scenario gradually changes from *z-cut first* to *z-cut last*. Therefore, we can use parameter α to control the number of TSVs required by a placement.

Next, we demonstrate how wirelength changes with parameter α and the size of TSV. To demonstrate how much benefit we can obtain from 3D technology, we normalize the 3-D wirelengths to the wirelengths of 2D placements which are produced by the same algorithm we proposed. For clarity, first we show the result of a single test case $IBM10$ in Figure 4.9. In the figure, the curves labeled 0×0, 5×5, 10×10 and 15×15 show the wirelengths using TSV sizes of $0\mu m$ by $0\mu m$, $5\mu m$ by $5\mu m$, $10\mu m$ by $10\mu m$, and $15\mu m$ by $15\mu m$, respectively. The circles indicate the optimal wirelengths. As shown, when the TSV size is $0\mu m$ by $0\mu m$, up to 23.0% of wirelength reduction can be gained with $\alpha = 0.4$. Therefore, when small TSV are used, *z-cut last* partitioning is beneficial for 3-D placement. It implies that a placer should defer the decision of vertical positions of cells when small TSV is applied. On the contrary, when large TSV is used, the optimal point shifts to *z-cut first* region with less value of α. It indicates that the 3-D design prefers to reduce the TSV overhead by first determining the tier which each cell belongs to.

Finally, we are interested in how much space we should reserve for TSVs. Assume the size of TSV is s. From the aforementioned analysis, we know that for each TSV size s, there is an optimal placement $PL_{opt}(s)$ with least wirelength. If the normalized wirelength of $PL(k_{opt})(s)$ is greater than 1, the wirelength of 3-D placement using a TSV size of s is no less than that of the 2-D placement. Also, we know that, the wirelength only increases with the increase of TSV size. Therefore, there must exist a threshold TSV size s_T such that when size of TSV $s > s_T$ the normalized wirelength of $PL(k_{opt})(s)$ is greater than one. We find the threshold

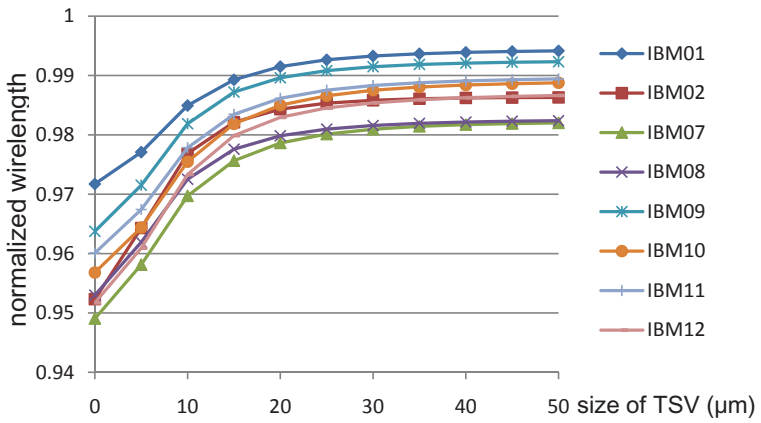
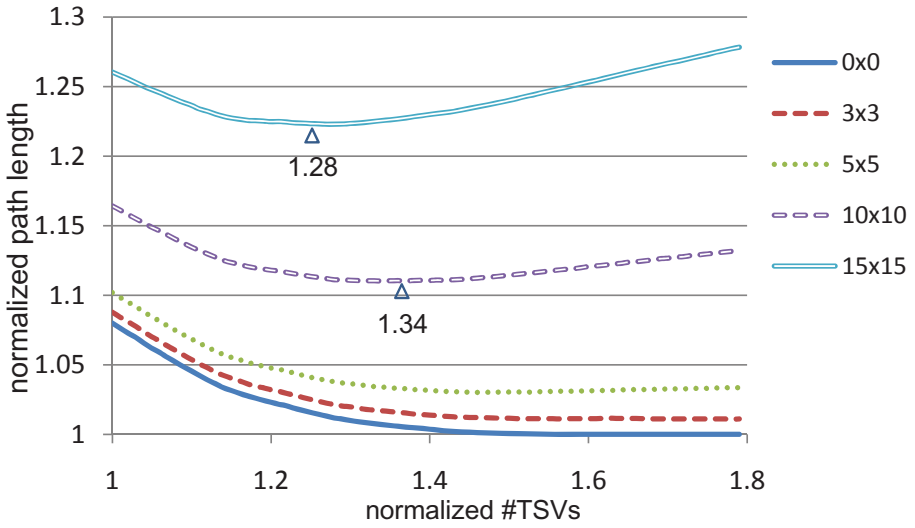


Figure 4.6: Normalized optimal wirelengths of all test cases.

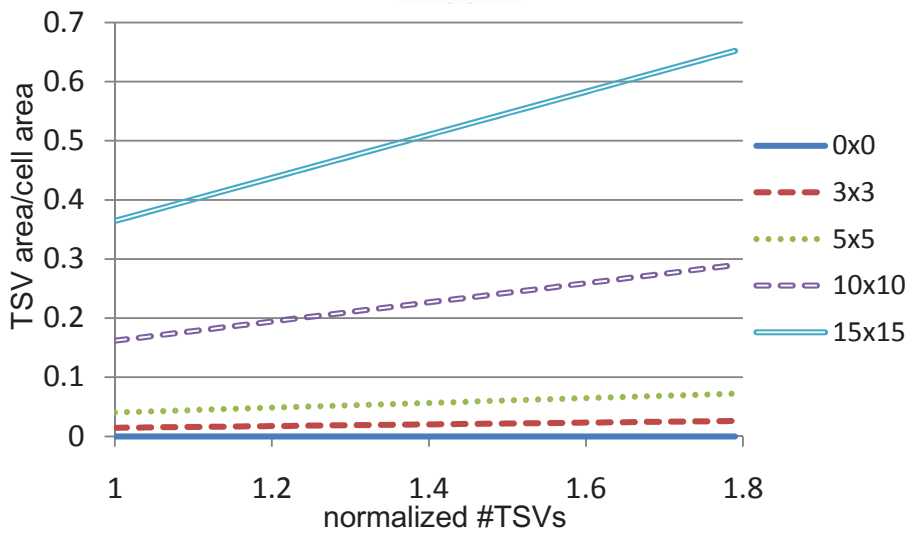
size s_T for all cases and compute the total TSV area of $PL_{opt}(s_t)$ by changing value of α from 0 to 1. The experimental results is given in Table 4.3. The columns labeled *circuit* and *TSV area* show the names of test cases and the maximum ratios of total TSV areas to the total cell areas, respectively. As the result shown, in our study cases, the average maximum TSV area should not excess 25.30% of the cell area; otherwise the wirelength will be longer than that of 2-D placements.

4.4 Summary

Although there is plenty of research discussing how wirelengths of 3D-ICs varies with the number of TSVs, most of their analyses are conducted without considering the TSV size or using only a single TSV size. In this chapter, we have studied how wirelengths of 3-D spanning trees change with number of TSVs under different TSV sizes. The analysis result reveals that, the optimal topologies of 3-D spanning trees is determined by the size of TSV. When the size of TSV is small,



(a) Length of longest path versus number of TSVs



(b) TSV area versus number of TSVs

Figure 4.7: TSV impact on longest paths.

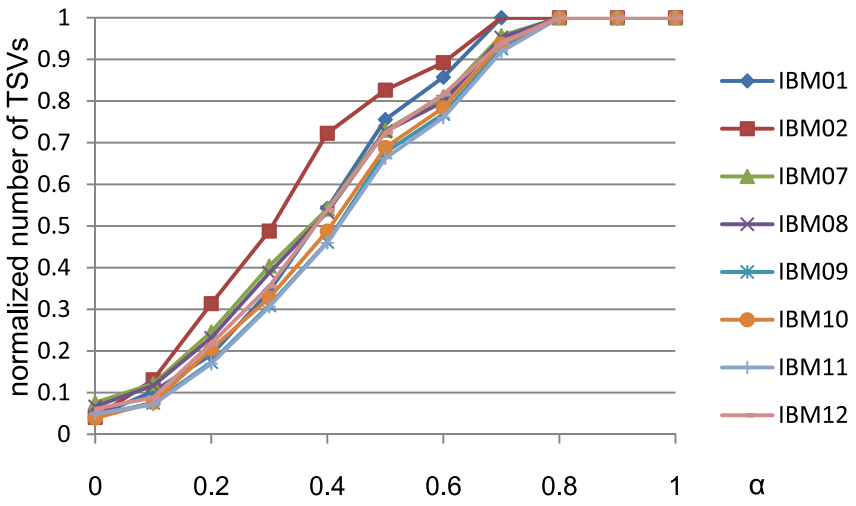


Figure 4.8: Number of TSVs and parameter α in our cutting scenario.

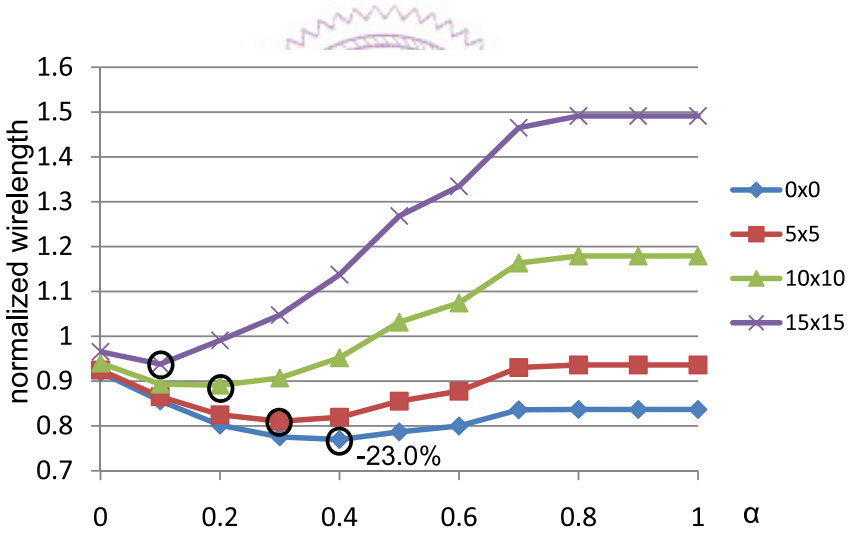


Figure 4.9: Normalized wirelength versus parameter α.

Table 4.3: Maximum TSV area ratio

circuit	*TSV area*	*circuit*	*TSV area*
IBM01	15.77%	IBM09	19.56%
IBM02	5.11%	IBM10	36.16%
IBM07	49.89%	IBM11	25.31%
IBM08	26.37%	IBM12	24.25%
avg TSV area		25.30%	

the using MSTs is beneficial for wirelength reduction. However, when large TSV is applied, a design prefers using spanning trees with least number of TSVs to minimize the total wirelength. Between these two extremes, a trade-off exists between wirelength and the number of TSVs. Moreover, we have also proposed a parameterized cutting scenario for partition-based placement and investigated how placement affects a 3-D design. Our experimental result shows that, the best partition scheme for placement depends on the TSV technology being applied. When large TSV are used, it is favorable to first partition circuit for assigning blocks to different tiers. On the contrary, the best placement is obtained by first determining the positions of blocks on the x-y plane and then performing tier assignment for blocks. Also, from our experimental result, we derived an upper bound for maximum total area of TSVs. Beyond this upper bound, virtually no wirelength reduction can be obtained. In our study cases, the average maximum ratio of total TSV area to total block area is 25.30%.

Chapter 5

Conclusion

In this dissertation, we have presented optimization methods for PSM and 3D integration technologies. First, we have proposed MILP-based wire spreading algorithms to modify layouts for PSM rules compliance. Different from the previous works which ask for finding a minimum weighted edge set or a minimal edge set whose removal makes the conflict graph 2-colorable, our algorithm use the exact deviation of wire segments as cost. The experimental results show the effectiveness of our algorithm; less than 2% of odd faces are left in the modified layout and no area increase is needed for the layout modification.

Second, we have proposed a 3D fixed-outline floorplanning algorithm to simultaneously place modules and plan TSVs for wirelength reduction. Compared to the average wirelength of a post-processing TSV placement algorithm, our result is shorter by 22.3%.In addition, we have also proposed a TSV re-assignment algorithm to refine a TSV planning result. Experimental results show that the wirelength can be reduced up to 11.39% without area overhead.

Third, we have studied how wirelengths of 3-D spanning trees change with

number of TSVs under different TSV sizes. Moreover, we have also proposed a parameterized cutting scenario for partition-based placement and investigated how placement affects a 3-D design. Our experimental result shows that, the best partition scheme for placement depends on the TSV technology being applied.

Although we have developed an MILP-based algorithm to modify the layout for PSM compliance, the runtime of the algorithm grows rapidly with the increase of problem size. Therefore, in the future, we are going to develop a heuristic algorithm based on divide-and-conquer approach to speedup the layout modification process. Furthermore, inspired by our evaluation results of 3D-IC, we also plane to develop an 3D placer which considers the TSV overhead and automatically decides the best number of TSVs for optimizing wirelength, timing or bandwidth of 3D-ICs.

fi

Bibliography

[1] http://www.reed-electronics.com/semiconductor/article/CA6356254

[2] Kevin McCullen, "Phase Correct Routing for Alternating Phase Shift Masks," *Proceedings of Design Automation Conference*, pp 317-320, 2004.

[3] P. Berman, A. B. Kahng, S. Mantik, I. L. Markov, and A. Zelikovsky,"Optimal Phase Conflict Removal for Layout of Dark Field Alternating Phase Shifting Masks,"*IEEE Transactions on CAD*, pp 1265-1278, 1999.

[4] K. Cao, J. Hu, and M. Cheng, "Layout Modification for Library Cell Alt-PSM Composablility," *Proceeding of International Society for Optical Engineering*, 2004.

[5] C. Chiang, A. B. Kahng S. Sinha and X. Xu, "Fast and Efficient Phase Conflict Detection and Correction in Standard-Cell Layouts," *Proceedings in International Conference on Computer Aided Design*, pp. 149-155, 2005.

[6] M. Chi and D.Z. Pan, “BoxRouter: A new Global Router Based on Box Expansion and Progressive ILP,” *Proceedings of Design Automation Conference*, pp. 373-378, 2006.

[7] http://www.silvaco.com

[8] D. Kral and H.-J. Vosss, “Edge-Disjoint Odd Cycle in Planar Graphs,” *Journal of Combinatorial Theory*, Series B, Vol 90, Issue 1, pp. 107-120, 2007.

[9] http://www.ilog.com

[10] K. McCullen, “Redundant Via Insertion in Restricted Topology Layouts,” *International Symposium on Quality Electronic Design*, pp. 821-828, 2007

[11] J. Cong, C. Liu, and G. Luo, “Quantitative Studies of Impact of 3D IC Design on Repeater Usage,” *Proceedings of International VLSI/ULSI Multilevel Interconnection Conference*, pp. 344-348, 2008.

[12] J. Burns, L. Mcllrath, C. Keast, et al., “Three-Dimensional Integrated Circuit for Low Power, High-Bandwidth Systems on a Chip,” *International Solid State Circuits Conference*, pp. 268-269, 2001.

[13] Y-J. Lee, Y. J. Kim, G. Huang, M. Bakir, Y. Joshi, A. Fedorov, and S. K. Lim, “Co-design of signal, power, and thermal distribution networks for 3D ICs,” *Design Automation & Test in Europe Conference*, pp.610-615, 2009.

[14] P. Garrou, C. Bower, and P. Ramm, “Handbook of 3D Integration: Technology an Applications of 3D Integrated Circuits,” John Wiley, New York, 2008.

[15] X. He, S. Dong, Y. Man, and X. Hong, “Simultaneous Buffer and Interlayer Via Planning for 3D Floorplanning,” *International Symposium on Quality Electronic Design*, pp. 740-745, 2009.

[16] J. Lu, S. Chen and T. Yoshimura, “Performance Maximized Interlayer Via Planning for 3D ICs,” *Proceedings of International Conference on ASIC*, pp. 1096-1099 2007.

[17] J. Cong, W. Jie, and Y. Zhang, “A Thermal-Driven Floorplan for 3D-ICs,” *Proceedings in International Conference on Computer Aided Design*, pp.306-313, 2004.

[18] Z. Li, X-L. Hong, Q. Zhou, S. Zeng, H. Yang, V. Pitchumani, and C-K. Cheng, “Integrating Dynamic Thermal Via Planning with 3D Floorplanning Algorithm,” *International Symposium on Physical Design*, pp.178-185, 2006.

[19] P-Q. Zhou, Y-C. Ma, Z-Y Li, R.P. Dick, S. Li, H. Zhou, X-L. Hong, and Q. Zhou, “3D-STAF: scalable temperature and leakage aware floorplanning for three-dimensional integrated circuits,” *Proceedings of International Conference on Computer-Aided Design*, pp.590-597, 2007.

[20] X. Li, Y. Ma, X. Hong, S. Dong, and J. Cong, "LP based white space redistribution for thermal via planning and performance optimization in 3D ICs," *Proceedings of Asia and South Pacific Design Automation Conference*, pp.209-212, 2008.

[21] B. Goplen, and S. Sapatnekar, "Placement of 3D ICs with Thermal and Interlayer Via Considerations," *Proceedings of Design Automation Conference*, pp. 626-631, 2007.

[22] J. Cong, G-J. Luo, J Wei, and Y. Zhang, "Thermal-Aware 3D IC Placement Via Transformation," *Proceedings of Asia and South Pacific Design Automation Conference*, pp.780-785, 2007.

[23] J. Li, and H. Miyashita, "Post-placement Thermal Via Planning for 3D Integrated Circuit," *Asia Pacific Conference on Circuits and Systems*, pp.808-811, 2006.

[24] J. Cong, and Y. Zhang, "Thermal Via Planning for 3-D ICs," *Proceedings of International Conference on Computer-Aided Design*, pp. 745-752, 2005.

[25] T. Zhang, Y. Zhan, and S. Sapatnekar, "Temperature-Aware Routing in 3D ICs," *Proceedings of Asia and South Pacific Design Automation Conference*, pp.308-314, 2006.

[26] http://www.magma-da.com/

[27] S.N. Adya, and I.L. Markov, “Fixed-Outline Floorplanning: Enabling Hierarchical Design,” *IEEE Transactions on Very Large Scale Integration Systems*, vol.11, no.6, pp. 1120-1135, 2003.

[28] T-C. Chen, Y-W. Chang, S-C. Lin, “IMF: Interconnect-Driven Multilevel Floorplanning for Large-Scale Building-Module Designs,” *Proceedings of International Conference on Computer-Aided Design*, pp.159-164, 2005.

[29] C. Chiang, S. Sinha, “The road to 3D EDA tool readiness, ” *Proceedings of Asia and South Pacific Design Automation Conference*, pp.429-436, 2009.

[30] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, “Rectangle Packing Based Module Placement,” *Proceedings of International Conference on Computer-Aided Design*, pp. 472-479, 1995.

[31] C-W. Sham, and E. F. Y. Young, “Routablility Driven Floorplanner with Buffer Block Planning,” *IEEE Transcation on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 22, Issue 4, pp 470-480, 2003.

[32] K. W. C. Wong, and E. F. Y. Young, “Fast Buffer Planning and Congestion Optimization in Interconnect-Driven Floorplanning,” *Proceedings of Asia and South Pacific Design Automation Conference*, pp 411-416, 2003.

[33] P. Sarkar, and C-K. Koh, “Routability-Driven Repeater Block Planning for Interconnect-Centric Floorplanning,” *IEEE Transcation on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 20, Issue 5, pp 660-671, 2001.

[34] http://www.algorithmic-solutions.com/leda/

[35] http://vlsicad.eecs.umich.edu/BK/parquet/

[36] P.S. Andry et al., "A CMOS-compatible Process for Fabricating Electrical Through-vias in Silicon," *Proceedings of Electronic Components and Technology Conference*, 2006, pp. 831-837

[37] S. Spiesshoefer, Z. Rahman, G. Vangara, S. Polamreddy, S. Burkett, and L. Schaper, "Process Integration for Through-Silicon Vias," *Journal of Vacuum Science & Technology A: Vacuum, Surfaces, and Films*, vol 23, issue 4, pp. 824-829, 2005.

[38] K. Takahashi and M. Sekiguchi, "Through Silicon Via and 3-D Wafer/Chip Stacking Technology," *Symposium on VLSI Circuits Digest of Technical Papers*, 2006.

[39] I. Loi, S. Mitra, T. H. Lee, S. Fujita, and L. Benini, "A Low-Overhead Fault Tolerance Scheme for TSV-Based 3D Network on Chip Links," *Proceedings of International Conference on Computer-Aided Design*, pp.598-602, 2008.

[40] J. A. Davis, V. K. De, and J. D. Meindl, "A Stochastic Wire-Length Distribution for Gigascale Integration (GSI)-Part I: Derivation and Validation," *IEEE Transcation on Electron Devices*, vol. 45, no. 3, pp. 580-589, 1998.

[41] J. W. Joyner, P. Zarkesh-Ha, J. A. Davis, and J. D. Meindl, "A Three-Dimensional Stochastic Wire-Length Distribution for Variable Separation of Strata," *Proceedings of Interconnect Technology Conference*, 2000.

[42] R. Zhang, K. Roy, C.-K. Koh, and D. B. Janes, "Stochastic Wire-Length and Delay Distributions of 3-Dimensional Circuits," *Proceedings of International Conference on Computer-Aided Design*, 2000.

[43] D. H. Kim, K. Athikulwongse, and S. K. Lim, "A Study of Through-Silicon-Via Impact on the 3D Stacked IC Layout," International Conference on Computer Aided Design, pp 674-680, 2009.

[44] M. Pathak, Y-J. Lee, T. Moon, S. K. Lim, "Through-Silicon-Via Management during 3D Physical Deign: When to Add and How Many?" International Conference on Computer Aided Design, pp 387-394, 2010.

[45] http://er.cs.ucla.edu/benchmarks/ibm-place/

[46] J. A. Roy, D. A. Papa, S. N. Adya, H. H. Chan, A. N. Ng, J. F. Lu, and I. L. Markov, "Capo: robust and scalable open-source min-cut floorplacer," Proceedings of International Symposium on Physical Design, 2005.

[47] N. Selvakkumaran and G. Karypis. Theto, "A Fast and High-Quality Paritioning Driven Global Placer," Technical Report 03-46, Department of Computer Science and Engineering, Univeristy of Minnesota, Novemver 2003.

[48] N. Selvakkumaran and G. Karypis. Theto, "A Fast and High-Quality Paritioning Driven Placement Tool," Technical Report 04-40, Department of Computer Science and Engineering, Univeristy of Minnesota, October 2004.

[49] http://glaros.dtc.umn.edu/gkhome/metis/hmetis/download

[50] http://er.cs.ucla.edu/benchmarks/ibm-place2/