Simultaneous Block and I/O Buffer Floorplanning for Flip-Chip Design*

Chih-Yang Peng[1], *Wen-Chang Chao*[1], *Yao-Wen Chang*[2], *and Jyh-Herng Wang*[3]

[1]Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 106, Taiwan

[2]Department of Electrical Engineering & Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 106, Taiwan

[3]Faraday Technology Corporation, Hsinchu 300, Taiwan

Abstract

The flip-chip package gives the highest chip density of any packaging method to support the pad-limited ASIC design. One of the most important characteristics of flip-chip designs is that the input/output buffers could be placed anywhere inside a chip. In this paper, we first introduce the floorplanning problem for the flip-chip design and formulate it as assigning the positions of input/output buffers and first-stage/last-stage blocks so that the path length between blocks and bump balls as well as the delay skew of the paths are simultaneously minimized. We then present a hierarchical method to solve the problem. We first cluster a block and its corresponding buffers to reduce the problem size. Then, we go into iterations of the alternating and interacting global optimization step and the partitioning step. The global optimization step places blocks based on simulated annealing using the B*-tree representation to minimize a given cost function. The partitioning step dissects the chip into two subregions, and the blocks are divided into two groups and are placed in respective subregions. The two steps repeat until each subregion contains at most a given number of blocks, defined by the ratio of the total block area to the chip area. At last, we refine the floorplan by perturbing blocks inside a subregion as well as in different subregions. Compared with the B*-tree based floorplanner alone, our method is more efficient and obtains significantly better results, with an average cost of only 51.8% of that obtained by using the B*-tree alone, based on a set of real industrial flip-chip designs provided by leading companies.

I. Introduction

A. Flip-chip Design

Flip-chip bonding gives the highest chip density of any packaging method to support the pad limited ASIC design. The important characteristics of flip-chip designs is that the signals or power could be imported from the signal bumps or power bumps distributed on the whole chip, and the input and output buffers could be placed anywhere inside a chip, like core cells. There exist a few different flip-chip architectures. See Figure 1 for an example layout of the flip-chip design available from UMC and ASE. We use the top metal or an extra metal layer, called *Re-Distributed Layer* (*RDL*), to connect input or output buffers to bump balls. Figure 2 illustrates the cross section of RDL. Bump balls are placed on RDL and use RDL to connect to IO buffers. Therefore, bump balls can overlap with input/output buffers and blocks.

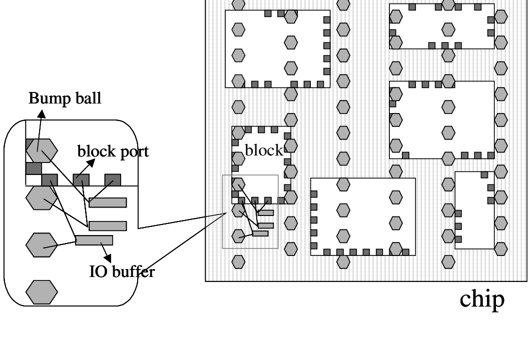

Fig. 1. Example layout of the flip chip.

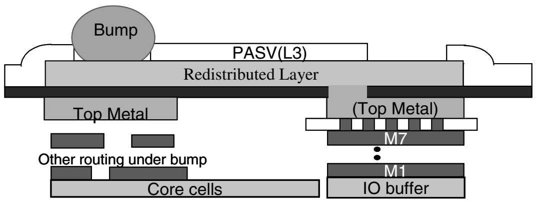

Fig. 2. Cross section of RDL.

In a wire-bond IC, in contrast, the circuit core is surrounded by the I/O pads on the perimeter of the chips. In general, the interconnection between an input/output gate and an I/O pad consists of two segments: the inner part and outer part. The inner-part segment is the portion of interconnection within the core while the outer part is between the core and the pads. Unlike the wire-bond I/O pads which are placed only on the perimeter of the chip, the flip-chip I/O pads are placed within the chip core. The inner-part routing can be minimized by placing the I/O pads. In the wire-bond layout, the area between the core and the pads is utilized for routing the interconnection of the outer-part segment. This area can be eliminated in a flip-chip IC by integrating the I/O pads within the core. This is a great saving in silicon area and generally occurs when there is a small core with a high number of I/O pins. The study conducted in [11] showed the reduction of die size and the increase in I/O count when the peripheral wire-bond technology was replaced by the flip-chip technology. Further, the bump balls in the flip chip have lower inductance than the bond-wires in the classical IC.

In this paper, we assume that all the bump balls are placed at pre-defined locations and their signals are determined, which is true for most real applications since they are often predefined by the packaging site. All core cells are partitioned or grouped into blocks. The input/output signals are connected to block ports through the input/output buffers to the bump balls. We need to place the input/output buffers and the blocks without overlapping with each other into a pre-defined chip area so that the path length between blocks and bump balls is minimized.

For most practical designs, like memory controllers, there are a large number of input/output pins being used as data buses. For such designs, we have to control the timing of the input/output signals. In other words, we have to make sure that the input signals arrive at the core simultaneously. In the same way, it also needs to make sure that the output signals arrive at bump balls simultaneously. This can be achieved through controlling the positions of bump balls, input/output buffers and blocks to minimize the signal skew.

B. Previous Work

The placement problem for the classical wire-bond IC's has been studied very extensively [1, 2, 5, 7, 8, 12, 13, 16, 17, 18, 22, 24, 25]. Nevertheless, most of these previous works target on standard cell designs, for which cells are of the same height and are placed in rows. For the floorplanning problem addressed here, it does not have such restrictions, making the previous works not flexible enough to the flip-chip floorplanning problem. (Note that although a few existing placers can handle the mixed-size placement problem, they usually focus more on the standard-cells of the same heights. Therefore, the placers cannot handle the floorplanning problem well. We have tried well-known publicly available placers such as Feng Shui 2.6/5.0 [13] and mGP [8, 22]. They all cannot obtain desirable floorplans for the flip-chip design directly.) Recently, Hsieh and Wang presented an analytical formulation for flip-chip placement in [15]. The work targets at an objective function of the sum of path delay and sum of skew between all input paths, which runs in quadratic time. Further, the sum of skew does not model the skew cost well.

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

In contrast, most floorplanning techniques can handle much more general objective functions by applying simulated annealing [9, 14, 21, 23, 26]. However, traditional floorplanning/placement algorithms do not scale well as the circuit size, complexity, and constraints increase. The B*-tree, in contrast, has been shown an efficient and effective data structure for floorplanning [6, 9, 20]. (In particular, the B*-tree tool is available on-line [4].) The B*-tree is particularly suitable for representing a floorplan/placement with mixed-size blocks, like the blocks and the I/O buffers for the flip-chip design; further, it does not have the cell height and the row placement constraints, imposed by the classical placement algorithms. Therefore, we shall take advantage of the nice properties of the B*-tree to develop our algorithm for flip-chip placement. Nevertheless, a key limitation of the B*-tree based floorplanner lies in its packing nature—the B*-tree based floorplanner always compacts blocks to the left and bottom as shown in Figure 3. If the total block area is much smaller than the chip area, then some blocks might not be placed at the desired positions to optimize the interconnection cost. As illustrated in Figure 3, the top and right sides of the chip are empty.

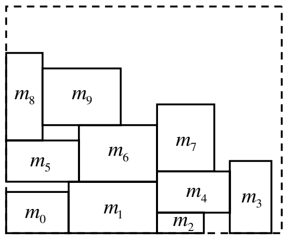

Fig. 3. The B*-tree based floorplanner always packs blocks to the left and to the bottom.

C. Our Contributions

In this paper, we first introduce the floorplanning problem for the flip-chip design and formulate it as assigning the positions of input/output buffers and first-stage/last-stage blocks so that the path length between blocks and bump balls as well as the delay skew of the paths are simultaneously minimized. In this formulation, we address practical issues in the industrial flip-chip design, such as the minimization of interconnection delay and data bus skew. To handle such objectives, the classical standard-cell/mixed-size placement techniques (like Aplace [16], Capo [1], GORDIAN [18], GORDIAN-L, mPG [8], mPL [7], FastPlace [24], Feng Shui [17], Dragon [25]) and the B*-tree floorplanning technique alone have their limitations. For example, the skew objective leads to a non-quadratic, non-convex term for which most analytic placers (such as Aplace, GORDIAN, mPL) rely on for the global optimization, the cell height and row placement constraints make the classical standard-cell placers not directly applicable to the flip-chip placement, and the compaction nature of the B*-tree limits the quality of interconnection optimization.

To remedy the limitations of the classical standard-cell/mixed-size placers and the B*-tree, we present a hierarchical top-down method to solve the problem based on a more accurate and efficient cost function. We first cluster a block and its corresponding buffers to reduce the problem size. Then, we go into iterations of the alternating and interacting global optimization step and the partitioning step. The global optimization step places blocks based on simulated annealing using the B*-tree representation to minimize a given cost function. The partitioning step dissects the chip into two subregions, and the blocks are divided into two groups and are placed in respective subregions. The two steps repeat until each subregion contains at most a given number of blocks, defined by the ratio of the total block area to the chip area. At last, we refine the floorplan by perturbing blocks inside a subregion as well as in different subregions. Compared with the B*-tree based floorplanner alone, our method obtains significantly better results, with an average cost of only 51.8% of that obtained by using the B*-tree alone, based on a set of real industrial flip-chip designs provided by leading companies. Further, our floorplanner is more efficient than the B*-tree alone.

The remainder of this paper is organized as follows. Section 2 formulates the problem of block and input/output buffer floorplanning for flip-chip design. Section 3 reviews the B*-tree representation. Section 4 presents our algorithm for handling the floorplanning problem. Section 5 reports the experimental results, and finally the conclusions and future work are given in Section 6.

II. Preliminaries

We consider the block-based design, for which blocks and buffers are rectangular and can be placed anywhere in the given flip-chip to minimize the objective function. All the input/output signals are connected to block ports through the input/output buffers. We assume that all the bump balls are placed at pre-defined locations (typically defined by packaging sites) and their signals are determined. We intend to minimize the interconnection length among the bump balls, the I/O buffers, and blocks. For most practical designs, as mentioned earlier, there are a large number of input/output pins being used for data buses. Therefore, it is also desired to make sure that all input signals from bump balls via input buffers to blocks arrive simultaneously and output signals from blocks via output buffers to bump balls also arrive simultaneously. To achieve the goal, we define the objective function Γ as follows:

$$\Gamma = \alpha\phi_1 + \beta\phi_2, \tag{1}$$

where

$$\phi_1 = \sum_{j=1}^{n1} d_j^i + \sum_{j=1}^{n2} d_j^o$$

$$\phi_2 = \left(\max_{1\le j\le n1} d_j^i - \min_{1\le j\le n1} d_j^i\right)^2 + \left(\max_{1\le j\le n2} d_j^o - \min_{1\le j\le n2} d_j^o\right)^2.$$

In Γ, ϕ_1 gives the sum of path delays, and ϕ_2 gives the sum of the squares of the maximum (critical) input and output signal skews. (Note that we adopt the squares of the signal skews in order to match the magnitude of the path delay cost.) Here, α and β are the user-specified weighting factors, $n1$ and $n2$ are the numbers of input and output signals, respectively, and d_j^i and d_j^o are the respective path delays of the jth input signal and the jth output signal. The path delay of an input signal is the delay of a path from a bump ball via an input buffer to a block port; the path delay of an output signal is the delay of a path from a block port via an output buffer to a bump ball. The path delay is measured by the rectilinear path length between two circuit components (bump balls, buffers, or block ports), i.e., the Manhattan distance between the two points. Minimizing the above objective function means that it needs to minimize the critical skew of the path delays of all input signals, output signals, and the total path delay.

It should be noted that the above objective function does address the requirements needed for the recent real industrial flip-chip designs (e.g., from the leading foundry UMC and its design service company Faraday) to be reported in Section 5. Also, unlike the objective function used in [15] which cannot model the skew cost accurately and needs quadratic time for evaluation, the new objective function is more accurate and needs only linear time for evaluation.

III. The B*-tree Representation

As mentioned earlier, we extend the B*-tree representation to handle the problem of block and I/O buffer floorplanning for flip-chip design. Thus, we shall give a review of the B*-tree representation.

Given a compacted placement P that can neither move down nor move left (called an *admissible placement* [14]), we can represent it by a unique B*-tree T [9]. (See Figure 4(b) for the B*-tree representing the placement shown in Figure 4(a).) A B*-tree is an ordered binary tree (a restriction of the O-tree [14] with faster and more flexible operations) whose root corresponds to the block on the bottom-left corner. Using the depth-first search (DFS) procedure, the B*-tree T for an admissible placement P can be constructed in a recursive fashion. Starting from the root, we first recursively construct the left subtree and then the right subtree. Let R_i denote the set of blocks located on the right-hand side and adjacent to m_i. The left child of the node n_i corresponds to the lowest block in R_i that is unvisited. The right child of n_i represents the lowest block located above m_i, with its x-coordinate equal to that of m_i.

Figure 4(b) illustrates the resulting B*-tree for the placement shown in Figure 4(a). The B*-tree keeps the geometric relationship between two blocks as follows. If node n_j is the left child of node n_i, block m_j must be located on the right-hand side and adjacent to block m_i in the admissible placement; i.e., $x_j = x_i + w_i$. Besides, if node n_j is the right child of n_i, block m_j must be located above block m_i, with the x-coordinate of m_j equal to that of m_i; i.e., $x_j = x_i$. Also, since the root of T represents the bottom-left block, the x- and y-coordinates of the block associated with the root $(x_{root}, y_{root}) = (0, 0)$. Therefore, given a B*-tree, the x-coordinates of all blocks can be determined by traversing the tree once. The y-coordinate can be computed based on the

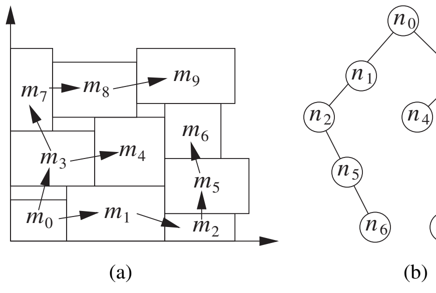

Fig. 4. (a) An admissible placement. (b) The corresponding B*-tree.

contour data structure presented in [14] in amortized $O(1)$ time for each node. Therefore, an n-node B*-tree can be evaluated very efficiently in amortized $O(n)$ time.

IV. Our Algorithm

Our algorithm is illustrated in Figure 5. Given inputs of net list and the geometry of the chip, we first cluster a block and its corresponding I/O buffers into a clustered block. Then we go into the main steps of alternating and interacting global optimization and partitioning steps. The global optimization step places blocks based on simulated annealing using the B*-tree representation to minimize a given cost function. The partitioning step dissects the chip into two subregions, and the blocks are divided into two groups according to their coordinates and are placed in respective subregions. Until each region contains at most q clustered blocks, we decluster these clustered blocks. After the declustering step, the global optimization and the partitioning steps repeat until each region contains at most k blocks. At last the final floorplanning step starts. In the final floorplanning step, we refine the floorplan by perturbing blocks inside a subregion as well as in different subregions. Note that the values of q and k control the resulting number of subregions. The smaller the values, the more the resulting subregions. If the area utilization ratio (the total block area divided by the total chip area) is large, it is harder to place blocks into subregions if we cut the chip into too many small subregions, and thus we shall favor larger q and k for this situation. We shall explain each step of the algorithm and the choices of q and k in the following sections.

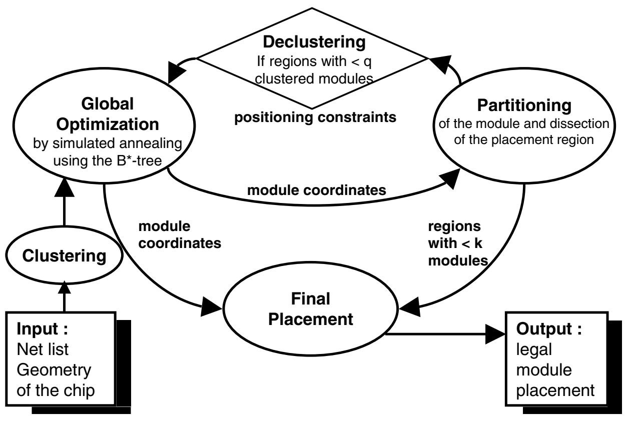

Fig. 5. Our algorithm.

A. Clustering

In this step, we apply simulated annealing using the B*-tree representation to group a block and its I/O buffers to a clustered block. The objective function is defined by area and the path delay between the input (output) port of a block and the output (input) port of an I/O buffer. By this process, I/O buffers will be clustered around its corresponding block. See Figure 6 for an example.

We introduce a node in the B*-tree based on the placement of a block and its clustered I/O buffers, called a *block node*. The width and height of the block node are equal to the respective width and height of the floorplan of the block and its clustered I/O buffers. When we perform the global optimization and partitioning, each block node represents its block and corresponding I/O buffers until declustering. Therefore, the problem size can significantly be reduced by clustering.

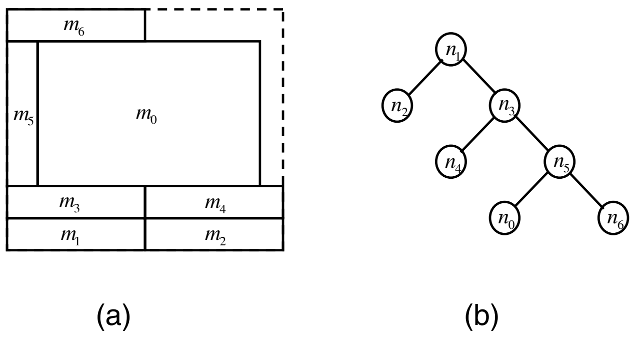

Fig. 6. (a) The geometry of a block and its clustered I/O buffers. (b) The B*-tree topology. m_0 is the block and $m_1, m_2, \cdots, m_n$ are the I/O buffers. The dotted line gives the boundary of the clustered block.

B. Global Optimization and Partitioning

The floorplanning procedure is composed of alternating and interacting global optimization and partitioning steps. In the global optimization step, we place blocks by simulated annealing using the B*-tree representation.

For the region ρ, the positions of all blocks in the region, denoted by M_ρ, are derived from simulated annealing using the B*-tree representation. We assume that W^r_ρ (H^r_ρ) is the width (height) of region ρ, and W^m_ρ (H^m_ρ) is the width (height) of the floorplan in region ρ. The width (W^r_ρ), height (H^r_ρ), and the coordinates of regions are determined in the partitioning step. The coordinate of each region is set to the bottom-left corner of the region, and the coordinates of blocks in M_ρ are relative to the coordinate of the region ρ.

There are two stages in the global optimization step, distinguished by the declustering step. For the global optimization before declustering, we place blocks only to minimize the objective function ϕ_1, and blocks might be placed out of the region at this stage. Here, ϕ_1 denotes the sum of wirelengths between the clustered blocks and bump balls. This process makes a clustered block closer to its corresponding bump balls. Although we do not consider the signal skew and the fixed outline of the flip chip at this stage, we can still fix/refine the solution at the final floorplanning stage or the global optimization step after declustering.

For the global optimization after declustering, we apply the objective function Γ. In order to place blocks into their region boundary (i.e., fixed-outline floorplanning), we shall also consider the width and height of the resulting floorplan individually so that neither dimension violates the outline constraint. To do so, we modify the objective function as follows:

$$\Gamma' \quad = \quad \Gamma + \gamma\Phi, \tag{2}$$

where

$$\Phi \quad = \quad \max(0, W^m_\rho - W^r_\rho) + \max(0, H^m_\rho - H^r_\rho). \tag{3}$$

Here, cost Φ is used to force blocks to be packed into the chip during simulated annealing. In order to satisfy the fixed-outline constraint, γ is set to a huge constant (say, 1000) to guarantee that the cost Φ is much bigger than any cost Γ if the fixed-outline constraint is violated.

After each global optimization process, a new partitioning step starts; we divide the region into two subregions and partition blocks into two groups depending on their positions. In the partitioning step, for each region ρ with $|M_\rho| > k$, if the region width (W^r_ρ) is larger than its height (H^r_ρ), the blocks in M_ρ are sorted according to the x-coordinates of the blocks, and the region is cut vertically. In contrast, if the region height is larger than its width, the blocks in M_ρ are sorted according to the y-coordinates, and the region is cut horizontally. Then, M_ρ is divided into $M_{\rho'}$ and $M_{\rho''}$ such that the summation of the block areas in $M_{\rho'}$ and $M_{\rho''}$ are approximately the same. The rectangular area of region ρ is dissected accordingly. See Figure 7 for an illustration of the processing.

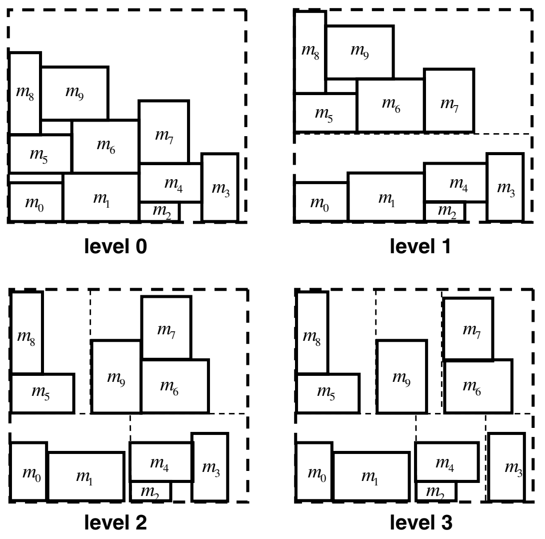

Fig. 7. Illustration of the interactive global optimization and partitioning steps ($q = 2$).

C. Declustering

If the number of blocks in any region is smaller than q (so the problem size is small enough), we shall ungroup each block which was formed by clustering a block and its corresponding I/O buffers previously in this region. Each node in the B*-tree is then expanded into a subtree representing the block's components which are constructed at the clustering step. The number of nodes after declustering is definitely larger. To cope with the increasing problem size, we process as follows to avoid a dramatic change in the resulting floorplan due to the declustering.

Suppose that the original tree has the nodes $n_0, n_1, \cdots, n_n$, denoting blocks $m_0, m_1, \cdots, m_n$, respectively; each block m_i corresponds to the subtree T_i constructed at the clustering step. There are two kinds of relations between two connected nodes in the original tree; a node is a left child or a right child of another node. Let n_l and n_r denote the left and right child of the node n_i; m_l and m_r represented by n_l and n_r are located right to or above m_i.

First we expand the parent node n_i into the subtree T_i and record the last contour when performing the B*-tree packing. The root node of the contour c_{root} represents the left- and top-most cell of the block m_i, and the tail node of the contour c_{tail} represents the right- and top-most cell. See Figure 8 for an illustration.

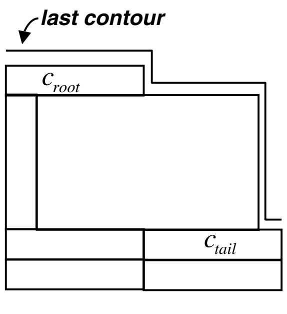

Fig. 8. The inter blocks and the last contour of the clustered block m_i.

Then, if the child of n_i is a left child n_l, we make the root node of the subtree T_l as the left child of the node c_{tail}. Thus the block represented by the root node of the subtree T_l is located adjacent to the right side of the right-most cell of block n_i, as illustrated in Figure 9. In contrast, if the child of n_i is a right child n_r, the root node of subtree T_l becomes the right child of the node c_{root}. The root cell will be located above the top-most cell of block n_i, as illustrated in Figure 10. By this process, the floorplan will not be changed dramatically after declustering.

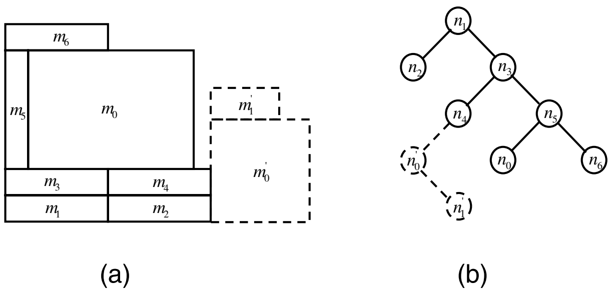

Fig. 9. The blocks of solid lines belong to a clustered block m_i, and the blocks of dotted lines belong to another clustered block m_l. m_l is the left child of m_i. (a) m_i and m_l after declustering. (b) The corresponding tree topology after declustering. (Here, m_4 is the tail bock on the last contour.)

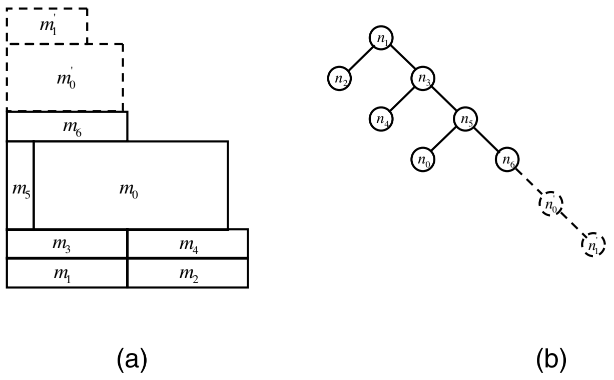

Fig. 10. The blocks of dotted lines belong to the clustered block m_r. m_r is the right child of m_i. (a) m_i and m_r after declustering. (b) The corresponding tree topology after declustering.

D. Final Floorplanning

In this step, the chip has been dissected into several subregions, and blocks have been divided into several groups and placed in respective subregions. The simulated annealing process starts again. We refine the floorplan by perturbing blocks inside a subregion as well as in different subregions. The objective function is the same as the function in the global optimization step, but the perturbation operations are different. We select two blocks randomly and swap them if the swap will not cause any outline violation. It gives a chance that a block can change its subregion. After changing blocks, we only re-compute the coordinates of blocks at the changed subregions. Doing so, we have a chance to further refine the floorplan solution.

E. Summary of Our Algorithm

The flow of our algorithm is shown in Figure 12 (the procedure is shown in Figure 5). The result of our floorplanning method may be influenced by the parameters q and k. The parameter q controls the degree of the partitioning step. A smaller q implies that more subregions will be partitioned. We suggest that if the total block area is much smaller than the chip area (i.e., the chip utilization ratio is small), the parameter q should be smaller to generate more subregions to prevent blocks from being packed together at the bottom-left corner of some subregion. (Note that this is an intrinsic behavior of a compacted floorplanner like the B*-tree.) Otherwise, we shall choose a larger q since it is harder to place blocks into subregions if we cut the chip into too many small subregions. The parameter k plays the same role as parameter q after declustering. The optimal q and k may be different for different test cases. Nevertheless, we propose a heuristic to define them based on the ratio of total block area to the chip area; the heuristic is given in Figure 11. It is clear that this heuristic leads to appropriate q and k for flip-chip design.

```
$q$ = # $clustered\_blocks$;
$k$ = # $blocks$;
$r = total\_blocks\_area/chip\_area$ /* utilization ratio */
if ( $r < 0.75$ )
    $q = 10 \times r$;
    $k = 10 \times r \times (\#blocks/\#clustered\_blocks)$
else
    $k = 10 \times r \times$ (# $blocks$ / # $clustered\_blocks$)
$q = \max\{q, 3\}$; /* the smallest $q = 3$ */
$k = \max\{k, 20\}$ /* the smallest $k = 20$ */;
```

Fig. 11. A heuristic to define the parameters q and k.

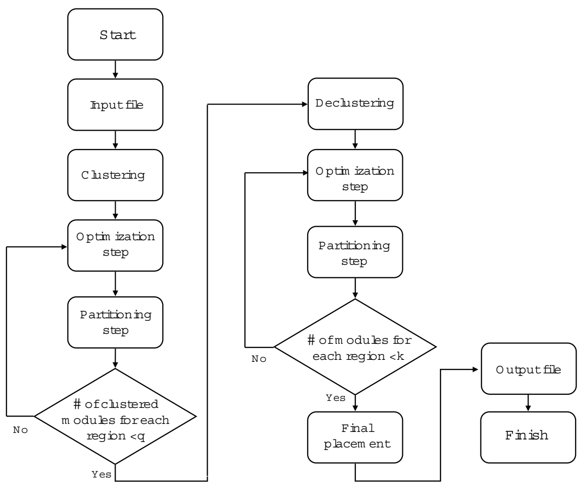

Fig. 12. The flow chart of our floorplanning method.

V. Experimental Results

We implemented our algorithm in the C++ programming language on a 1.2GHz SUN Blade 2000 workstation with 8 GB memory. (We will make this tool available to the public after this work is published to facilitate future research along this direction.) The benchmark circuits fc1, fc2, . . ., fc7 are real consumer designs (DVD players, MP3, etc) and were provided by the leading foundry UMC and its design service company Faraday. Table I lists the names of circuits, the number of blocks, the number of buffers, the chip areas, the ratio of the total blocks area (including blocks and I/O buffer blocks) to the chip area, and the parameters α and β (also defined by the company). The parameter α is the weighting factor of the path delay part ϕ_1 of the objective function Γ, and the β is that of the skew part ϕ_2 of Γ. The test cases fc4, fc5, fc6, and fc7 are for the same design with different assignments of block ports and wire connections; therefore, their chip sizes and the block sizes are all the same. (So the problem sizes range from 31 blocks + I/O buffers to 412 blocks + I/O buffers, representing the typical problem sizes for recent applications.)

We compared our algorithm with the state-of-the-art B*-tree floorplanner and the TCG [21] one using the same cost function Γ. It should be noted that we do not compare with the classical standard-cell placers. As mentioned earlier, the classical standard-cell and/or mixed-size placers (such as the famous Aplace, Capo, GORDIAN, GORDIAN-L, mPG, mPL, FastPlace, Feng Shui, Dragon) cannot directly apply to the flip-chip floorplanning problem well because of the cell height and row placement constraints and the non-quadratic, non-convex term in the problem formulation. (As mentioned earlier, we have tried well-known publicly available placers such as Feng Shui 2.6/5.0 [13] and mGP [8, 22]. They all cannot obtain desirable floorplans for the flip-chip design directly.) We shall also note that the B*-tree floorplanner is considered a leading tool in block floorplanning [6, 10, 20].

The results are listed in Table II. The B*-tree package that we used here is the state-of-the-art version used in [20], which has been shown to be able to handle up to thousands of blocks. The source code of the B*-tree package is available to the general public on-line [4]. We implemented our algorithm based on the same simulated annealing scheme as that used by the B*-tree package. Unlike the B*-tree based floorplanner which always compacts blocks to the left and bottom, the TCG based floorplanner results in general floorplans, which well addresses the layout requirement for the flip-chip design. Nevertheless, TCG has a higher complexity of $O(n^2)$ time for its operations and packing with n blocks. This limits the applicability and quality of TCG for large-scale designs.

As shown in Tables II and III, our method obtains significantly better results in total path delays and the input/output signal skews; the B*-tree based algorithm (the TCG based algorithm) results in the overall cost of 2.04 times (1.52 times) of that of our algorithm. Note that because of the higher complexity in operations and packing, the TCG based floorplanner alone is only feasible for the first two cases. Further, our method is more efficient than the B*-tree and the TCG-based floorplanners. The results justify the effectiveness and efficiency of our method; the B*-tree based algorithm (the TCG based algorithm) needs 1.28 times (more than 332 times) of our CPU time. The results show the effectiveness and efficiency of our algorithm. The resulting layout fc3 is shown in Figures 13.

It should be noted that the reason why our method is even more efficient than the B*-tree based floorplanner mainly lies in the hierarchical framework and the clustering and declustering schemes adopted in our work. By using the framework and the schemes, we can control the problem sizes well at each

Circuit	# blocks	# buffers	chip area	block area /chip area	α	β
fc1	6	25	1040x1040	0.4216	0.5	0.5
fc2	12	168	3440x3440	0.5598	0.5	0.5
fc3	23	320	4240x4240	0.6584	0.7	0.3
fc4	28	384	4440x4440	0.7276	0.7	0.3
fc5	28	384	4440x4440	0.7276	0.7	0.3
fc6	28	384	4040x4040	0.8788	0.7	0.3
fc7	28	384	4040x4040	0.8788	0.7	0.3

TABLE I
Statistics of the test circuits.

Ckt		B*-tree alone		TCG alone		Our Method	
fc1	Tot. path delay	23390	1.32	28430	1.60	17760	1.0
	Max. input skew	160	1.33	120	1.00	120	1.0
	Max. output skew	100	1.11	100	1.11	90	1.0
	Cost Γ	2.95e+06	1.46	2.641e+06	1.31	2.01e+06	1.0
	CPU Time	1 s	0.73	32 s	33.83	1 s	1.0
fc2	Tot. path delay	521030	1.44	750450	2.08	361650	1.0
	Max. input skew	1360	1.37	1390	1.38	1010	1.0
	Max. output skew	1890	1.36	1740	1.25	1390	1.0
	Cost Γ	2.97e+08	1.79	2.855e+08	1.72	1.66e+08	1.0
	CPU Time	20 s	1.29	9944 s	631.76	16 s	1.0
fc3	Tot. path delay	1033800	1.67	NR	-	619200	1.0
	Max. input skew	3320	2.00	NR	-	1660	1.0
	Max. output skew	2500	1.47	NR	-	1700	1.0
	Cost Γ	1.24e+09	3.00	NR	-	4.14e+08	1.0
	CPU Time	85 s	1.66	$>$10 hr	-	51 s	1.0
fc4	Tot. path delay	1153560	1.59	NR	-	726040	1.0
	Max. input skew	3380	1.54	NR	-	2190	1.0
	Max. output skew	2820	1.18	NR	-	2380	1.0
	Cost Γ	1.39e+09	1.84	NR	-	7.54e+08	1.0
	CPU Time	130 s	1.80	$>$10 hr	-	72 s	1.0
fc5	Tot. path delay	969140	1.37	NR	-	707430	1.0
	Max. input skew	3300	1.91	NR	-	1730	1.0
	Max. output skew	3200	1.48	NR	-	2160	1.0
	Cost Γ	1.51e+09	2.71	NR	-	5.57e+08	1.0
	CPU Time	130 s	1.66	$>$10 hr	-	78 s	1.0
fc6	Tot. path delay	1233720	1.65	NR	-	745880	1.0
	Max. input skew	3580	1.19	NR	-	3000	1.0
	Max. output skew	4360	1.39	NR	-	3140	1.0
	Cost Γ	2.26e+09	1.69	NR	-	1.34e+09	1.0
	CPU Time	108 s	0.68	$>$10 hr	-	160 s	1.0
fc7	Tot. path delay	1159560	1.59	NR	-	729180	1.0
	Max. input skew	3880	1.11	NR	-	3500	1.0
	Max. output skew	4720	1.65	NR	-	2860	1.0
	Cost Γ	2.65e+09	1.82	NR	-	1.45e+09	1.0
	CPU Time	251 s	1.11	$>$10 hr	-	226 s	1.0

TABLE II
Experimental results of our floorplanning method, the B*-tree representation alone and TCG representation alone.
***NR:** no results obtained.

	Total path delays	Max. input skew	Max. output skew	Cost Γ	CPU Time
Our method	1.00	1.00	1.00	1.00	1.00
B*-tree alone	1.52	1.49	1.38	2.04	1.28
TCG alone	1.84	1.19	1.18	1.52	332.80

TABLE III
THE AVERAGE COST AND CPU TIME RATIOS FOR THE B*-TREE AND THE TCG BASED ALGORITHMS VS. OUR ALGORITHM FOR ALL TEST CIRCUITS.

stage. Therefore, our method has better scalability than the B*-tree and TCG-based floorplanners for handling the flip-chip floorplanning of various problem sizes.

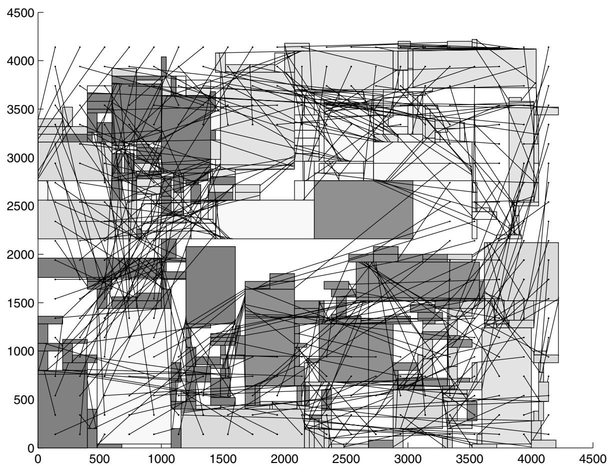
Fig. 13. The floorplanning result of fc3.

VI. CONCLUSION

We have presented a B*-tree based hierarchical top-down method for the block and input/output buffer floorplanning for flip-chip design. This method not only remedies the limitations of the classical standard-cell placers and the B*-tree, but also speeds up the running time by applying the hierarchical top-down scheme. Experimental results based on real industrial flip-chip designs provided by leading companies have shown the effectiveness and efficiency of our algorithm. Future work lies in developing other heuristics to slice the chip to further improve the results. Also, the routing and tighter integration of layout and packaging co-synthesis for the flip-chip design are on-going.

REFERENCES

[1] S. N. Adya, I. L. Markov, and P. G. Villarrubia, "On whitespace in mixed-size placement and physical synthesis," *Proc. of IEEE/ACM Int. Conf. on Computer-Aided Design*, pp. 311–318, 2003.

[2] A. R. Agnihotri, M. C. Yildiz, A. Khatkhate, A. Mathur, S. Ono, and P. H. Madden, "Fractical cut: improved recursive bisection placement," *Proc. of IEEE/ACM Int. Conf. on Computer-Aided Design*, pp. 307–310, 2003.

[3] P.H. Buffet, J. Natonio, R.A. Proctor, Yu H. Sun, and G. Yasar, "Methodology for I/O cell placement and checking in ASIC designs using area-array power grid," *Proc. of IEEE Custom Integrated Circuits Conf.*, pp. 125–128, 2000.

[4] B*-tree: http://cc.ee.ntu.edu.tw/∼ywchang/research.html.

[5] A. E. Caldwell, A. B. Kahng, and I. L. Markov, "Can recursive bisection alone produce routable placement?," *Proc. of ACM/IEEE Design Automation Conf.*, pp. 477–482, 2000.

[6] H. H. Chan, S. N. Adya, and I. L. Markov, "Are floorplan representations important in digital design?" *Proc. of ACM International Symposium on Physical Design*, pp. 129–136, 2005.

[7] T. Chan, J. Cong, and K. Sze, "Multilevel generalized force-directed method for circuit placement," *Proc. of ACM International Symposium on Physical Design*, 2005.

[8] C. C. Chang, J. Cong, and X. Yuan, "Multilevel placement for large-scale mixed-size IC designs," *Proc. of ACM/IEEE Asia and South Pacific Design Automation Conf.*, pp. 325–330, 2003.

[9] Y.-C. Chang, Y.-W. Chang, G.-M. Wu, and S.-W. Wu, "B*-Trees: a new representation for non-slicing floorplans," *Proc. of ACM/IEEE Design Automation Conf.*, pp. 458–463, 2000.

[10] J. Con, G. Nataneli, M. Romesis, and J. R. Shinnerl, "An area-optimality study of flooprlanning," *Proc. of ACM International Symposium on Physical Design*, pp. 78–83, April 2004.

[11] P. Dehkordi and D. Bouldin, "Design for packageability: the impact of bonding technology on the size and layout of VLSI dies," *Proc. Multichip Module Conf.*, pp. 153–159, 1993.

[12] H. Eisenmann and F. M. Johannes, "Generic global placement and floorplanning," *Proc. of ACM/IEEE Design Automation Conf.*, pp. 269–274, 1998.

[13] *FengShui Placer*. http://vlsicad.cs.binghamton.edu/software.html.

[14] P.-N. Guo, C.-K. Cheng, and T. Yoshimura, "An O-tree representation of non-slicing floorplan and its applications," *Proc. of ACM/IEEE Design Automation Conf.*, pp. 268–273, 1999.

[15] H.-Y. Hsieh and T.-C. Wang, Simple yet effective algorithms for block and I/O buffer placement in flip-chip designs, *Proc. of IEEE International Symposium on Circuits and Systems*, pp. 1879–1882, May 2005.

[16] A. B. Kahng and Q. Wang, "Implementation and extensibility of an analytic placer," *Proc. of ACM International Symposium on Physical Design*, pp. 18–25, April 2004.

[17] A. Khatkhate, C. Li, A. R. Agnihotri, M. C. Yildiz, S. Ono, C.-K. Koh, and P. H. Madden, "Recursive bisection based mixed block placement," *Proc. of ACM International Symposium on Physical Design*, pp. 84–89, April 2004.

[18] J.M. Kleinhans, G. Sigl, F.M. Johannes, K.J. Antreich, "GORDIAN: VLSI placement by quadratic programming and slicing optimization," *IEEE Trans. Computer-Aided Design*, pp. 356–365, 1991.

[19] J.N. Kozhaya, S.R. Nassif, F.N. Najm, "I/O buffer placement methodology for ASICs," *Proc. of IEEE International Conference on Electronics, Circuits and System*, pp. 245–248, 2001.

[20] H.-C. Lee, Y.-W. Chang, J.-M. Hsu, and H. Yang, "Multilevel floorplanning/placement for large-scale modules using B*-trees," *Proc. of ACM/IEEE Design Automation Conf.*, Anaheim, CA, June 2003.

[21] J.-M. Lin and Y.-W. Chang, "TCG: A transitive closure graph based representation for non-slicing floorplans," *Proc. of ACM/IEEE Design Automation Conf.*, pp. 764–769, Las Vegas, NV, June 2001.

[22] *mGP: Multilevel Global Placement*. http://ballade.cs.ucla.edu/mGP/.

[23] Parquet: Fixed-Outline Floorplanner, http://vlsicad.eecs.umich.edu/BK/parquet/

[24] N. Viswanathan and C. C.-N. Chu, "FastPlace: efficient analytical placement using cell shifting, iterative local refinement and a hybrid net model," *Proc. of ACM International Symposium on Physical Design*, pp. 26–33, April 2004.

[25] M. Wang, X. Yang, and M. Sarrafzadeh, "Dragon2000: standard-cell placement tool for large industry circuits," *Proc. of IEEE/ACM Int. Conf. on Computer-Aided Design*, pp. 260–263, June 2000.

[26] H. Zhou and J. Wang, "ACG–Adjacent constraint graph for general floorplans," *Proc. of IEEE Int. Conf. on Computer Design*, pp. 572–575, October 2004.