

Block Alignment in 3D Floorplan Using Layered TCG

Jill H.Y. Law
Department of CSE
Chinese University of Hong
Kong

hylaw@cse.cuhk.edu.hk

Evangeline F.Y. Young
Department of CSE
Chinese University of Hong
Kong

fyyoung@cse.cuhk.edu.hk

Royce L.S. Ching
Department of CSE
Chinese University of Hong
Kong

lsching@cse.cuhk.edu.hk

ABSTRACT

In modern IC design, the number of long on-chip wires has been growing rapidly because of the increasing circuit complexity. Interconnect delay has dominated over gate delay as technology advances into the deep submicron era. 3D chip is a feasible solution to these problems. It has been shown that interconnect lengths can be greatly reduced in 3D ICs. In this paper, a novel 3D floorplan representation namely Layered Transitive Closure Graph (LTCG) is proposed, which is based on the Transitive Closure Graph (TCG) representation for 2D non-slicing floorplans. In LTCG, we can impose topological relationships between both blocks of the same layer and blocks of different layers. Experimental results have shown that LTCG is very promising for multi-layer floorplanning and can handle the inter-layer alignment problem effectively.

Categories and Subject Descriptors

B.7.2 [Integrated Circuits]: Design aids—*Placement and routing*; J.6 [Computer Applications]: Computer-aided design—*Computer-aided design (CAD)*

General Terms

Algorithm, Design

Keywords

3D floorplanning, Block alignment

1. INTRODUCTION

Interconnect optimization has become a major concern in modern floorplanning. Interconnect delay has dominated over gate delay and timing constraints have become more and more difficult to be met with this huge number of interconnects involved. 3D chip is a feasible solution to these problems. It has been shown that interconnect lengths can be greatly reduced in 3D ICs. A 3D chip is actually a chip with more than one silicon layers to place modules. Traditional 3D representations cannot be used directly here to solve the 3D floorplanning problem as

the modules are actually not 3-dimensional. The authors of [1] proposed a slicing tree structure representation for multi-layer floorplans. Similar to 2D floorplan, a slicing tree is constructed for 3D floorplan. There are three kinds of internal nodes, ‘H’, ‘V’, and ‘Z’ representing horizontal, vertical, and lateral cuts respectively, and each leaf is labelled by a module name. To realize a floorplan, the slicing tree has to be broken down. Each layer is represented by a slicing sub-tree. This is done by removing all the ‘Z’ nodes in the tree, leaving behind those ‘V’ and ‘H’ nodes only. To make things simple, some researchers have proposed to use an array of 2D representations to represent 3D floorplans [2] [3]. In [2], the authors proposed to use an array of Bounded-Sliceline Grid (BSG) to represent a 3D floorplan, where each BSG represents the 2D floorplan on a layer. In [3], the same approach is used, but sequence pair is used as the 2D floorplan representation. This kind of representations is simple but the relationship between blocks in different layers cannot be represented.

The authors of [4] proposed a multi-layer representation called Combined Bucket and 2D Array (CBA). CBA is consisted of two parts, an array of 2D representation to represent the packing on each layer, and a bucket structure to store the relationships between blocks on different layers. In [4], Transitive Closure Graph (TCG) is used as the 2D representation but in fact, any 2D floorplan representation, like Sequence Pair or Corner Block List, can be used. A bucket represents a rectangular region on the x - y plane. Each bucket stores the set of blocks (on different layers) whose projection on the x - y plane intersects with the rectangular region represented by that bucket. Besides, for each block, the indexes of the buckets it is stored in are recorded. Thus, if two blocks i and j , locating on different layers, are stored in the same bucket, it is likely that they are placed close to each other. This representation stores the relationships between blocks on different layer but a lot of unused information may be stored. The runtime of the floorplanner will be lengthened in order to update those unused information. Besides, even if two blocks i and j are stored in the same bucket, they are just close to each other and we cannot control their relative positions directly.

In this paper, a novel 3D floorplan representation namely Layered Transitive Closure Graph (LTCG) is proposed, which is based on the TCG representation for 2D non-slicing floorplans. In LTCG, we can impose topological relationships between both blocks of the same layer and blocks of different layers. This property can be made used of to address different issues, like aligning blocks on the same bus, reducing interconnect length, controlling the relative positions between some inter-layer blocks to reduce temperature, etc. We have devel-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

GLSVLSI'06, April 30–May 2, 2006, Philadelphia, PA, USA.

Copyright 2006 ACM 1-59593-347-6/06/0004 ...\$5.00.

oped a 3D-floorplanner based on LTCG that can align blocks on different layer by adding edges in the constraints graphs to address the bus-driven floorplanning problem in 3D floorplan design specifically. Experimental results have shown that LTCG is very promising for 3D floorplanning and can handle the inter-layer alignment problem effectively.

The rest of the paper is organized as follows. A 3D floorplan representation, LTCG, will be proposed in Section 2. Our 3D floorplanner based on LTCG that can align blocks vertically will be discussed in Section 3, followed by the experimental results in Section 5. Finally, a conclusion will be drawn in Section 6.

2. LAYERED TRANSITIVE CLOSUR GRAPH (LTCG)

Based on TCG, we proposed a 3D floorplan representation called Layered Transitive Closure Graph (LTCG). The representation is consisted of two main components: the layer information and the transitive closure graphs. The layer information stores the layer assignment of each block. The two transitive closure graphs show the topological relationship between the blocks.

We denote the two transitive closure graphs in LTCG by $G_h = (V, E_h)$ and $G_v = (V, E_v)$ where V is a set of vertices to represent the blocks. For modules on the same layer, an edge must exist between every pair of them in either G_h or G_v . For pairs of blocks located on different layers, they may or may not have topological relationship with each other. Thus, an edge may exist between them in either G_h or G_v , but there can also be no such edges.

Similar to TCG, LTCG have three properties:

1. G_h and G_v are acyclic.
2. Each pair of nodes i and j , where i and j are assigned to the same layer, must be connected by exactly one edge in G_h or G_v .
3. Let $G_h^k = (V_k, E_h^k)$ ($G_v^k = (V_k, E_v^k)$), where $1 \leq k \leq K$ and K is the number of layers, be a sub-graph of $G_h = (V, E_h)$ ($G_v = (V, E_v)$), such that V_k is a set of vertices representing the blocks on layer k , $E_h^k \subseteq E_h$ and E_h^k contains only those edges with both end points in V_k . For $1 \leq k \leq K$, the transitive closures of G_h^k and G_v^k are equal to themselves respectively.

The floorplan realization process can also be done by performing a longest path search for each node in G_h and G_v , which takes $O(n^2)$ time. An example of using LTCG to represent a layered floorplan is shown in Figure 1.

3. BLOCK ALIGNMENT IN 3D FLOORPLANS USING LTCG

3.1 Problem Formulation

We define the problem of 3D floorplanning with block alignment as follows:

Problem: 3D Floorplanning with Block Alignment Given a set of n modules $M = \{M_1, M_2, \dots, M_n\}$ and a value K that represents the number of layers and m sets of modules $U = \{u_0, u_1, \dots, u_{m-1}\}$. Each module M_i is associated with an area A_i and two aspect ratio bounds r_i and s_i such that $r_i \leq h_i/w_i \leq s_i$, where h_i and w_i are the height and width

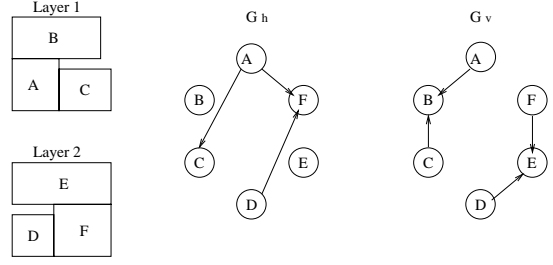


Figure 1: A Multi-Layer Floorplan and Its LTCG Representation.

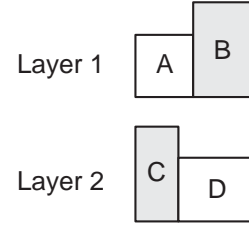


Figure 2: Block A, D and Block B, C cannot be aligned simultaneously.

of module i respectively. Each u_i is a set of blocks B_i where $B_i \subseteq M$. We want to find a feasible 3D floorplan F , i.e., the coordinates (x_i, y_i) and the dimensions (h_i, w_i) of modules i , and the layer l_i on which module i lies, where $1 \leq l_i \leq K$, such that if the blocks in B_i where $1 \leq i < m$ are placed in k different layers $l_{i,\pi(1)}, l_{i,\pi(2)}, \dots, l_{i,\pi(k)}$ where $2 \leq k \leq K$ ($l_{i,\pi(1)} < l_{i,\pi(2)} < \dots < l_{i,\pi(k)}$), there is at least one block on layer $l_{i,\pi(j)}$ aligning with a block on layer $l_{i,\pi(j+1)}$ in the z -direction for $1 \leq j \leq k - 1$.

3.2 Aligning Blocks

There are two graphs, G_h and G_v , in LTCG to govern the horizontal and vertical relationships between the blocks. If we want two blocks X and Y located on different layers to align in the z -direction, the blocks have to overlap vertically and horizontally. To achieve this, a pair of edges (X, Y) and (Y, X) , both with zero weight, can be added into the graphs.

For simplicity, we assume that there are only two layers for placing blocks in the following discussion. Consider a $u_i \in U$. Let $P = \{p_1, p_2, \dots, p_a\}$ be the set of blocks in u_i and on the first layer and $Q = \{q_1, q_2, \dots, q_b\}$ be the set of blocks in u_i and on the second layer. Our task is to find a block p_i from P and a block q_j from Q such that p_i and q_j can align in the z -direction. In some cases, it is not possible to do alignment for all the u_i s. An example is shown in Figure 2 in which block A and D and block B and C cannot be aligned simultaneously. We want to select a pair of blocks from each u_i that has blocks lying on two different layers to be aligned in the z -direction such that the number of aligned u_i s is maximized. The case of having K layers, where $K > 2$, can be considered as having $k - 1$ sub-problems of two layers and a similar approach can be applied.

To select a pair of blocks from each u_i to be aligned, we will scan one of the two graphs G_h or G_v first. In our floorplanner, we will consider the graph G_h first. Our proposed method is consisted of several iterations. In each iteration, vertices in G_h that have no incoming edges and is not in any u_i are

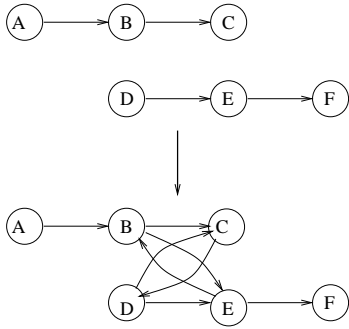


Figure 3: Cycle Exists if The Two Pair of Edges are Added Simultaneously.

first removed. Then a set S of candidate vertices are found, where each of them has no incoming edges in G_h and belongs to some u_i s. Then pairs of vertices in S that are on different layers and belong to the same u_i are matched. After matching, the matched vertices are removed from G_h . When no more matchings can be done, a vertex is randomly selected from S and removed from G_h . The whole process is repeated until for each u_i , one pair of vertices are matched, or until G_h is empty.

Note that matching candidate vertices in a topological order as described above can avoid creating positive cycles in G_h . An example is shown in Figure 3. Suppose block B and E are on different layers and belong to the same u_i , and block C and D are on different layers and belong to the same u_j . Adding pairs of edges between B and E and between C and D simultaneously will yield a positive cycle. In our approach, we will only match B and E or C and D depending on whether B or D is randomly selected and deleted from G_h .

Suppose two blocks X and Y are selected to be matched for u_i . It is guaranteed that no positive cycles will be produced in G_h after inserting a pair of zero weighted edges in G_h because none of the two blocks is a predecessor of the other in G_h . However, we also need to add those pair of edges (X, Y) and (Y, X) in G_v and cycle may be formed in G_v . If adding a pair of edges to G_v will produce a positive cycle, the pair will not be matched and the corresponding edges will not be added. The pseudo code of the procedure to align blocks is shown in Figure 4.

4. OUR 3D FLOORPLANNER

Simulated annealing is used to search for a good solution. A set of moves are designed to change one candidate solution to another. LTCG is capable for handling block alignment effectively in 3D floorplans. We can align blocks on different layers by adding edges into the LTCG. Details of our 3D floorplanner will be presented in the following.

4.1 Cost Function

Our objective is to minimize the area of the floorplan and the number of unaligned u_i s. Thus, the cost function is defined as follows:

$$Cost = \alpha \cdot A + \beta \cdot I$$

where A is the chip area and I is the number of unaligned u_i s. α and β are parameters that can be specified by the users. Though we aim at minimizing the area of the floorplan and the number of unaligned u_i s, other aspects like total wire length

ALIGN_BLOCK

```

01 align ← 0
02 FOR i from 0 to m
03   matched[i] ← FALSE
04 ENDFOR
05 WHILE ( $G_h$  not empty) & (align < m) /*start iteration*/
06   Remove vertices with no incoming edges and not in any  $u_i$ 
07    $S$  ← vertices with no incoming edges
08   FOR i from 0 to m - 1
09     FOR all pairs  $x, y \in S$  in  $u_i$  and on different layers
10       IF matched[i] = FALSE
11         IF adding  $(x, y)$  &  $(y, x)$  yields no cycles in  $G_v$ 
12           add  $(x, y)$  &  $(y, x)$  in  $G_h, G_v$  with weight 0
13           align ++
14           delete vertex  $x$  and  $y$  in  $G_h$ 
15            $S$  ←  $S - \{x, y\}$ 
16           matched[i] ← TRUE
17         END IF
18       END IF
19     END FOR
20   END FOR
21   IF no matching is done in this iteration
22      $k$  ← randomly select a vertex in  $S$ 
23     delete vertex  $k$  in  $G_h$ 
24   END IF
25 END WHILE

```

Figure 4: Pseudo Code of Aligning Blocks.

and routing congestion can be taken into account by including more terms in the cost function.

4.2 Solution Perturbation

To change from one candidate solution to another, we have defined several types of moves: *rotate*, *swap*, *move*, *reverse*, *remove*, *add*, and *change-layer*. Details of each of them will be discussed in the following.

4.2.1 Rotate

In this operation, a randomly selected module is rotated. Rotating a module means interchanging the width and height of the module.

4.2.2 Swap

In this operation, two randomly selected modules are swapped. To swap two modules x and y , the corresponding nodes in both G_h and G_v will be exchanged.

4.2.3 Move

To move an edge means moving an intra-layer edge from either G_h or G_v to the other graph. A *reduction edge* (x, y) (an edge (x, y) is said to be a reduction edge if there exists no other paths from block x to block y in the same graph) is first selected randomly from G_h or G_v , where x and y are on the *same* layer. Then, the edge is moved to the other graph. This will change the relationship between x and y from horizontal (vertical) to vertical (horizontal).

To maintain the properties of LTCG, some checkings have to be performed after the move. Assume that (x, y) is moved from G to G' . After the move, for each node $n_i \in F_{in}(x) \cup \{x\}$ in G' and $n_j \in F_{out}(y) \cup \{y\}$ in G' , we need to check whether (n_i, n_j) exists in G' . If not, it should be added to G' and the corresponding edge in G should be removed.

Note that after applying the move operation, no cycles will be created. It can be proved by contradiction. Assuming that a cycle exists after adding (x, y) to G' . That cycle must involve the edge (x, y) as the original graph is acyclic. This means that there exists a path from y to x in G' . However, according to property 3 of LTCG, if a path exists from y to x in G' , an edge (y, x) will also exist in G' , contradictory to the fact that (x, y) is an edge in G . Therefore, after the move operation, the graphs will remain acyclic, and the transitive closure property will also be preserved.

4.2.4 Reverse

Reverse means reversing the direction of a randomly selected reduction edge (x, y) in G_h or G_v , where x and y are on the same layer. This operation will also change the geometric relationship between the blocks. To reverse an edge (x, y) in G ($= G_h$ or G_v), it is deleted from G first and then the edge (y, x) is added back to G .

Similar to the “Move” operation, checkings have to be performed to maintain the properties of LTCG. For each node $n_i \in F_{in}(y) \cup \{y\}$ in G and $n_j \in F_{out}(x) \cup \{x\}$ in G , we need to check whether (n_i, n_j) exists. If not, it should be added to G and the corresponding edge in the other graph (G_v or G_h) should be deleted.

After reversing an edge, the acyclic property and the transitive closure property will also be preserved. The latter is maintained by performing the checkings as described above. The former can be proved by contradiction. Assume that a cycle exists after reversing an edge (x, y) . It means that a path exists from x to y originally. This contradicts to the fact that (x, y) is a reduction edge in G (there exists no other paths from x to y). Therefore, it is guaranteed that the properties of LTCG are maintained after the move.

4.2.5 Remove

In LTCG, blocks on different layers may bear one or no relationship with each other. In this operation, the relationship between a pair of randomly selected blocks on different layers is removed.

4.2.6 Add

This operation is the opposite of remove operation. The add operation adds an edge between a pair of randomly selected blocks in G_h or G_v , where the blocks belong to different layers. Note that after adding the edge, the graph should remain acyclic. If adding a selected edge (x, y) yields a cycle, the edge will not be added.

4.2.7 Change-Layer

In this operation, a randomly selected block x is moved from one layer to another. It will be placed on the boundary of the destination layer. For every block y that is no longer on the same layer as x , both (x, y) and (y, x) will be removed from G_h and G_v . For every block z that is now on the same layer as x , edges (x, z) are added in C_h if x is chosen to be placed on the leftmost boundary of the destination layer. Similarly, if x is chosen to be placed on the rightmost boundary, the bottommost boundary, or the topmost boundary, corresponding edges have to be added in the closure graphs.

4.3 Soft Block Adjustment

After placing the modules as hard blocks, soft block adjustment is done to change the shapes of the blocks. This is again

Table 1: Characteristics of Data Set 1.

Benchmark	No. of Blocks	No. of Layers
ami33	33	4
ami49	33	4

Table 2: Comparisons between [1] and LTCG.

Benchmark	Layer No.	Deadspace of [1]	Deadspace of LTCG	Runtime (s)
ami33	4	3.09%	1.95%	13
ami49	4	3.76%	2.49%	28

done by another simulated annealing process. In each perturbation, a block is selected randomly and its shape is changed a little bit, as long as the aspect ratio bounds of the block is not violated. The cost function is defined as follows:

$$cost = \alpha \times A + \beta \times S$$

where A is the total area of the floorplan and S is the difference between the current aspect ratio bound and the required aspect ratio bound of the floorplan. This soft block adjustment step is shown to be essential by the experimental results as the floorplan utilization can be greatly increased.

5. EXPERIMENTAL RESULTS

A 3D floorplanner was implemented with the C++ language, using the LTCG representation. All the experiments were conducted in an Intel P4 (3.2 GHz) machine with 2G memory. The MCNC benchmarks and the GSRC benchmarks were used. We have conducted two sets of experiments. The first set of experiments just perform 3D floorplanning without block alignment. The characteristics of the benchmarks used (Data Set 1) are shown in Table 1. The results are shown in Table 2. Comparisons showed that our floorplanner can out-perform the floorplanner proposed in [1]. As the runtime was not reported in [1], only the runtime of our floorplanner is shown in Table 2. For all the experiments, the best of twenty trials are reported. The runtime reported is the average of the twenty trials.

The second set of experiments considers block alignment. The sets of blocks u_i are randomly constructed in the benchmarks. We have constructed two sets of data (Data Set 2 and Data Set 3). The characteristics of Data Set 2 are shown in Table 3. In this data set, m is large, though the number of blocks in each u_i is small. The characteristics of Data Set 3 are shown in Table 4. In this data set, m is smaller, but the number of blocks involved in each u_i is large. Experimental results after soft block adjustment are shown in Table 5 and Table 6. All floorplans are packed successfully with every alignment constraint satisfied. The deadspace is 5.86% on average for Data Set 2, and 4.97% on average for Data Set 3. Experimental results showed that LTCG is very promising for multi-layer floorplanning and can handle inter-layer block alignment very effectively.

6. SUMMARY AND CONCLUSIONS

As the complexity of VLSI circuit design increases, the number of interconnects involved has grown rapidly. 3D chips can reduce interconnect lengths significantly. We have proposed a 3D floorplan representation namely *Layered Transitively Closure Graph* (LTCG), based on the Transitive Closure Graph [5] representation for non-slicing floorplans. Based on LTCG, we proposed a method to align blocks on different layers, by adding

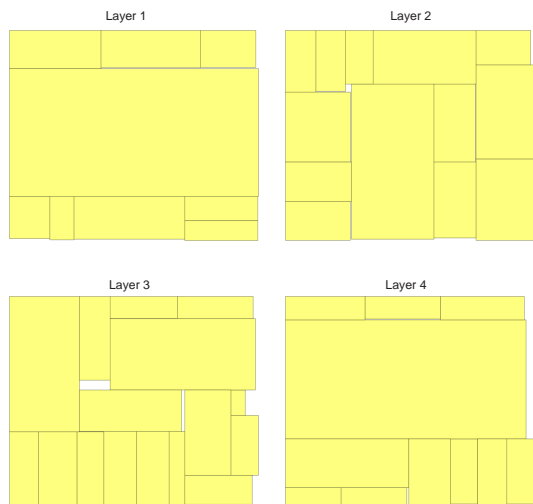


Figure 5: One result packing of ami49 in Data Set 1.

Table 3: Characteristics of Data Set 2.

Benchmark	No. of Blocks	No. of Layers	No. of u_i s	Avg./Max. No. of Blocks in a u_i
apte	9	2	3	2/2
hp	10	2	3	2/2
xerox	11	2	3	2/2
ami33	33	3	7	2.29/3
ami49	49	4	7	2.29/3
n100a	100	4	10	2.21/3

Table 4: Characteristics of Data Set 3.

Benchmark	No. of Blocks	No. of Layers	No. of u_i s	Avg./Max. No. of Blocks in a u_i
apte	9	2	2	4/4
hp	10	2	1	6/6
xerox	11	2	2	4.5/5
ami33	33	3	2	7.5/8
ami49	49	4	3	7/9
n100a	100	4	5	7.4/10

Table 5: Experimental Results of Data Set 2.

Benchmark	Time(s)	Deadspace
apte	2	2.06%
hp	4	8.18%
xerox	4	5.47%
ami33	23	5.32%
ami49	89	6.30%
n100a	371	7.84%

Table 6: Experimental Results of Data Set 3.

Benchmark	Time(s)	Deadspace
apte	4	1.77%
hp	3	8.44%
xerox	7	3.16%
ami33	28	5.24 %
ami49	63	4.10%
n100a	545	7.13%

pairs of edges in LTCCG. A floorplanner was implemented using the LTCCG representation, and the experimental results are very promising.



Figure 6: A result packing of ami49 in Data set 3 ($B_1 = \{0-5, 32, 33, 44\}$, $B_2 = \{6-11\}$, $B_3 = \{12-17\}$).

7. REFERENCES

- [1] J. Berntsson and M. Tang, "A Slicing Structure Representation for the Multi-Layer Floorplan Layout Problem," in *European Workshop on Evolutionary Computation in Hardware Optimization*, 2004.
- [2] Yangdong Deng and W. P. Maly, "Interconnect Characteristics of 2.5-D System Integration Scheme," in *Proceedings of the 2001 International Symposium on Physical Design*, pages 171-175, 2001.
- [3] P. H. Shiu, R. Ravichandran, S. easwar, and S. K. Lim, "Multi-Layer Floorplanning for Reliable System-on-Package," in *Proceedings of the 2004 IEEE International Symposium on Circuits and Systems*, 2004.
- [4] J. Cong, J. Wei, and Y. Zhang, "A Thermal-Driven Floorplanning Algorithm," in *Proceedings of the 2004 International Conference on Computer Aided Design*, 2004.
- [5] J. M. Lin and Y. W. Chang, "TCG: A Transitive Closure Graph-Based Representation for Non-Slicing Floorplans," in *Proceedings of the 38th Conference on Design Automation*, 2001.
- [6] K. Bazargan and R. Kastner and M. Sarrafzadeh, "3-D Floorplanning: Simulated Annealing and Greedy Placement Methods for Reconfigurable Computing Systems," in *Design Automation for Embedded Systems (DAfES) - RSP'99 Special Issue*, 2000.