Through Via Generation and Placement for 3D Stacked ICs

ABSTRACT

Through-via is the key enabling technology that allows stacking and interconnecting multiple die together in 3D stacked ICs. This paper formulates and provides effective solutions to the two new problems related to the through-via management during the physical design phase of 3D stacked ICs: constrained min/max partitioning and 3D Steiner routing. In case the given design needs to be partitioned for multiple-die implementation, all inter-partition connections require through-vias. Our constrained min/max partitioning is formulated in such a way that various die bonding styles available in 3D stacked IC are exploited. Once the gates placed, we construct performance-oriented 3D Steiner tree for each net while determining the optimal location for the related through-vias. Our related experiments show the effectiveness of our proposed solutions.

1. INTRODUCTION

Technology scaling has resulted in fabrication of smaller and faster devices. The smaller device size has resulted in higher device density and greater number of interconnects. In addition, technology scaling caused interconnect delay to play a dominating factor in today's designs. Various optimization techniques like wire sizing/spacing, buffer insertion, and wire shielding have been developed to overcome the interconnect problem. However, these techniques increase silicon area and power consumption. To solve the issues related to interconnect, new design processes need to be looked into. 3-dimensional integration is an effective design paradigm for interconnect-centric circuits. The ability to route signals in the vertical dimension enables distant blocks to be placed on top of each other. This results in a decrease in the overall wire length, which translates into less wire delay, less power, and greater performance. Although the extra degree of freedom is an attractive option, the physical design problems pose interesting challenges.

The wafer-bonding approach in [1] joins discrete wafers using a copper interconnect interface, and permits multiple wafers and multiple 3D interconnects. Different bonding styles can be used for the fabrication of 3-D stacked ICs, which can be broadly classified into face-to-face (F2F), face-to-back (F2B), and back-to-back (B2B) bonding styles, as illustrated in Figure 1. The "face" refers to the metal layer side of a die, whereas the substrate side is called "back". *Through-vias* refer to the vias connecting die in these bonding styles and thus are the key enabling technology for 3D die stacking. This paper focuses on the effective management of the through vias during the physical design for 3D stacked ICs. We note that two major steps in physical design have high impact on through via management: partitioning and routing. In case the given design needs to be partitioned for multiple-die implementation, all inter-partition connections require through-vias. Therefore, we view the die-partitioning step as the one that *generates* through vias. In addition, it is during the routing step that these through vias are actually *placed*.

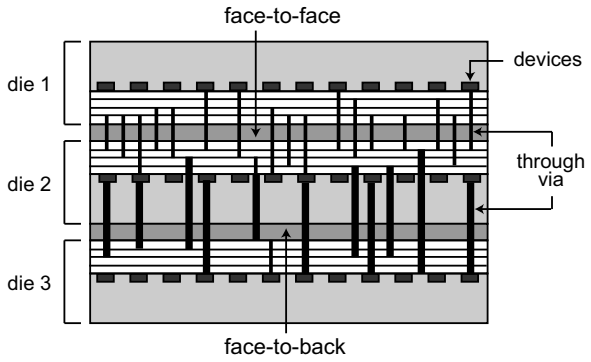

Figure 1: Bonding Styles in 3D stacked ICs

F2F through-vias ($\approx 0.5\mu \times 0.5\mu$) have a smaller pitch than F2B and B2B through-vias ($\approx 5\mu \times 5\mu$) [2]. In addition, too many F2B/B2B through-vias fabricated on a single thinned wafer may adversely affect reliability [3] since these vias actually penetrate the substrate. Thus, it is desirable to *decrease* the number of inter-die connections in these bonding styles. In the case of F2F bonding, however, it is desirable to *increase* the number of inter-die connections since the via density is much higher and enables a significantly higher bandwidth for inter-layer communication. On the other hand, the location of through vias in a Steiner tree has high impact on the overall topology as well as the delay at the sink nodes of the tree since it determines the amount of wiring done at all intermediate die that the tree spans. To the best of our knowledge, none of these problems have been addressed in the past.[1] Thus, the contributions of this paper are as follows:

DAC'06, July 24–28, 2006, San Francisco, California, USA.

[1]Some analytical results on the location of through vias were recently presented in [4]. However, this work focused on two-pin nets and did not perform Steiner tree construction.

- We formulate and solve the new *Constrained Min/max Partitioning* problem for bonding style-aware through-via generation in 3D stacked ICs. Our performance-driven multi-level partitioning approach recursively partitions the netlist while alternating the objectives between mincut and maxcut under cutsize constraints.
- We formulate and solve the new *3D Steiner Routing* problem for multi-pin net routing in 3D stacked ICs. Our algorithm considers all die and their bonding styles simultaneously and constructs performance-oriented Steiner trees while determining the optimal location for the related through-vias.

The remainder of the paper is organized as follows. Section 2 presents the problem formulation. Section 3 and 4 respectively presents our 3D partitioning and 3D routing algorithms. Experimental results are presented in Section 5, and we conclude in Section 6.

2. PROBLEM FORMULATION

2.1 Constrained Min/Max Partitioning

We assume the following is given: (i) a netlist represented by a directed graph $G = (V, E)$, where the area and delay of each gate $v \in V$ are given, (ii) an ordered set of K die $\{D_0, D_1, \cdots, D_{K-1}\}$, (iii) bonding style for each pair of adjacent die, denoted $bond(D_i, D_{i+1}) \rightarrow \{F2F, F2B, B2B\}$, (iv) cutsize upper bound for each pair of adjacent die $cut(D_i, D_{i+1}) < C_i$, and (v) area constraint for each partition $A_{low} < |D_i| < A_{up}$. The goal of the *Constrained Min/Max Partitioning* problem is to partition G into K die so that the cutsize and area constraints are satisfied. The objective is to minimize the critical path delay, where the delay of interconnect $e \in E$ is based on its length. In general, $C_{F2F} \gg C_{F2B}$ and $C_{F2B} \simeq C_{B2B}$, i.e., the cutsize constraint for F2F is much higher than F2B and B2B.

2.2 3D Steiner Routing

We assume the following is given: (i) a set of m nets $\{n_0, n_1, \cdots, n_{m-1}\}$, where each net is represented by a list of pins $n_i = \{p_0, p_1, \cdots, p_{k-1}\}$ with p_0 as the driver, (iii) a $X \times Y \times Z$ 3D routing grid R that represents the routing resource in a given 3D stacked IC, where each grid node represents a routing region and each edge denotes the adjacency among the regions, (iv) each x/y grid edge is associated with horizontal/vertical wire capacity and z with via capacity, and (v) the location of each pin $p(x, y, z)$. A *3D Steiner Tree* is defined to be a set of 2D (= planar) Steiner trees connected by through vias. The goal of the *3D Steiner Routing* problem is to generate a 3D Steiner tree for each net while satisfying the capacity constraints specified in the underlying R. The objective is to minimize the maximum Elmore delay among all pins in each tree.

2.3 Design Flow

Our physical design flow for 3D stacked ICs is as follows:

1. 3D Partitioning: we partition the netlist into multiple die. Our constrained min/max partitioning is used for this step.
2. 3D Placement: the gates in each die are placed. We extended the existing simulated annealing-based detailed placement [5] to 3D for this step.
3. 3D Routing: we construct Seiner tree for each net. Our 3D Steiner routing is used for this step.

3D floorplanning [6] and placement [7] are well studied. Thus, the focus of this paper is the first and the third step.

Constrained Min/Max Partitioning Algorithm
input: netlist NL, area/cutsize constraints, K (= # of die)
output: ordered set of K partitions of NL

```
1.  while (# of partition ≠ K)
2.      P = next partition in BFS order;
3.      if (bond(P0, P1) → F2F)
4.          C_P = max-cut multi-level clustering of P;
5.      else
6.          C_P = min-cut multi-level clustering of P;
7.      hgt = height of C_P;
8.      B(hgt) = random bipartitioning of P at top level;
9.      for (h = hgt downto 0)
10.         refine B(h) based on bond(P0, P1);
11.         project B(h) to B(h − 1);
```

Figure 2: Pseudocode of our recursive multi-level layer partitioning algorithm. The clustering and partitioning objectives are alternated between maxcut and mincut depending on the bonding style.

3. LAYER PARTITIONING ALGORITHM

3.1 Overview of the Algorithm

Our bonding-aware layer partitioning algorithm is based on recursive multi-level bipartitioning approach. A pseudocode is shown in Figure 2. Given a netlist, the algorithm recursively bipartitions the netlist until the desired number of partitions is obtained (line 1-2). Each cutline is optimized by iteratively improving the random initial solution, where groups of gates (= clusters) are moved across the cutline to improve the solution quality (line 10). Multi-level paradigm is based on the fact that the size of the clusters being moved is gradually reduced so that the finer and finer-grained optimization is carried out to effectively improve the solution (line 9-11). Multi-level approach requires a multi-level cluster hierarchy, which is obtained by recursively clustering the netlist. Once the refinement at a certain cluster-level is completed, i.e., no further improvement is possible, the next lower-level clusters are introduced and refinement starts with the current solution (line 11).

Upon each bipartitioning, the clustering and partitioning objectives are set according to the bonding style that it corresponds to (line 3, 10). In case of F2F bonding, the cutsize constraint is usually set high due to the high density of F2F through vias available. We first perform recursive clustering so that the amount of connection among the clusters is *maximized* (line 4). Next, we generate a random bipartitioning, which in most case has a lower cutsize than the target cutsize (line 8). Then the subsequent multi-level refinement attempts to *increase* the cutsize until it hits the cutsize target. Then the subsequent refinement tries to maintain the cutsize while improving the performance objective until all levels of the cluster hierarchy are examined. In case of F2B/B2B bonding, the cutsize constraint is usually set low due to the low density of F2B/B2B through vias available. Our multi-level clustering tries to *minimize* the amount of connection among the clusters (line 6). Then the multi-level partitioning refinement is performed to *decrease* the initial cutsize until it hits the target. The complexity of our partitioning algorithm is $O(n \log n)$ as the traditional multi-level partitioning algorithm since the maxcut clustering or maxgain computation do not increase the complexity.

3.2 Min/Max Clustering and Partitioning

We discuss the following three important issues regarding our layer partitioning: min/max clustering, min/max gain computation,

and timing optimization under cutsize constraint. In case of F2F bonding, our recursive clustering attempts to maximize the amount of interconnect among the clusters. This is because it improves the chance of finding a maxcut (cutline with high cutsize). Thus, our greedy maxcut clustering heuristic tries to group clusters with fewer connections among them. We first model the netlist with an edge-weighted undirected graph, where each net of size k forms a k-clique with edge weight of $1/(k-1)$. We then visit a node x and find another node y so that there is no connection between x and y or the weight of $e(x, y)$ is minimum. We then cluster x and y if the resulting cluster size does not exceed the limit. This greedy heuristic runs in $O(n^2)$ time in the worst due to the all-pair comparison. However, the runtime in practice is much faster since searching for a non-connected partner does not usually require scanning all other nodes due to the sparseness of the graph. In case of F2B/B2B bonding, any known clustering method for cutsize minimization can be used such as edge coarsening (EC) [8].

In general, the cluster move in an iterative improvement-based partitioning is guided by the *gain* concept. The traditional gain of a cluster in mincut bipartitioning denotes the cutsize reduction if it is moved to the other partition. In our layer partitioning algorithm, the gain under maxcut objective is recomputed so that it represents the *increase* in cutsize if moved. Thus, the two opposite definitions of the gain are used based on the partitioning objective. During a single *pass* of an iterative improvement-based partitioning scheme, all clusters are given a chance to move. We note that it is often possible to hit the high/low cutsize target while the current pass is not completed. In this case, we use the remaining moves of the current pass to optimize timing objective while maintaining the cutsize close to the target. Before each pass starts, we perform static timing analysis to compute the timing weight of each net. We then minimize/maximize the timing-weighted cutsize during the remaining moves. In this case, we do not allow the move of any cluster that causes the real cutsize to deviate too much from the target. In case of F2F bonding, placing two gates on critical path *on top of* each other may help timing better then placing them *next to* each other. This is the reason for *maximizing* the weighted cutsize and thus separating the gates into two dies for timing optimization in F2F bonding.

4. 3D STEINER ROUTING ALGORITHM

4.1 Overview of the Algorithm

The basic approach of our 3D Steiner tree construction algorithm is similar to SERT [9], where we incrementally grow the existing tree by connecting a new pin to it. The goal is to minimize the maximum Elmore delay among all sink pins of the tree. Here the biggest challenge is to compute the point on the tree where the new pin connects to. The major difference between SERT and our work is that all the pins in SERT are located in the same die, whereas our 3D algorithm handles the pins located in different die. This 3D case requires the usage of through vias, and the location of these vias has huge impact on the topology of the tree as well as the sink pin delay. In addition, the delay optimization in SERT is based on single variable, whereas our algorithm deals with two-variable function optimization.

A pseudocode of our algorithm is shown in Figure 3. Our routing algorithm consists of two phases: construction (line 1-13) and refinement (14-15). We construct 3D Steiner trees during the construction phase while ignoring congestion and then alleviate congestion by rip-up-and-reroute during the refinement phase. Given a net n, our 3D Steiner tree T_n initially contains the driver pin (line 2). We store the remaining pins of n in Q_n (line 3). We

3D Steiner Routing Algorithm
input: placed netlist NL, routing resource graph R
output: 3D Steiner tree for each net

```
1.  for (each net n ∈ NL)
2.      T_n = p_0(n);
3.      Q_n = set of pins of n except p_0;
4.      while (Q_n ≠ ∅)
5.          for (each pin a ∈ Q_n)
6.              for (each edge e ∈ T_n)
7.                  x = optimal connection point for a → e;
8.                  y = optimal through via location on e(x, a);
9.                  update d(p) for all p ∈ T_n using T_n ∪ e(x, a);
10.                 X(a, e) = max{d(p)|p ∈ T_n ∪ a};
11.         (a_min, e_min) = pin+edge pair with min X;
12.         T_n = T_n ∪ e_min;
13.         remove a_min from Q_n;
14. for (each non-timing critical T_n that violates capacity)
15.     rip-up-and-reroute T_n;
```

Figure 3: Pseudocode of performance-driven 3D Steiner routing algorithm. In case e and a are located in different planes, $e(r, a)$ will contain a through via.

then examine all pin-edge pair (line 5-6) and compute the impact of connecting the pin to the edge on Elmore delay, where the pin is from Q_n and the edge is from T_n. Specifically, the delay impact is calculated based on the increase in maximum Elmore delay among all pins currently in T_n (line 9-10). This requires the computation of connection point x and the through via location y (line 7-8) (to be discussed in Section 4.3). Next, we select the pin-edge pair that results in the minimum max-delay increase (line 11) and add the pin to T_n (line 12-13). Our rip-up-and-reroute is done only on the less timing critical nets, i.e., the nets with smaller max-delay values (line 14-15). The complexity of our algorithm is $O(nm^4)$, where n is the total number of nets and m is the maximum number of pins in any net. The $O(m^4)$ term is based on the while-loop, two for-loops, and Elmore delay computation for all sinks (line 9).

4.2 Elmore Routing for 3D Steiner Trees

In this section we discuss how we can efficiently construct 3D Steiner trees. Our discussion is based on two-die case for the simplicity of the discussion, but our algorithm is applicable to multiple die without any modification. Let r_1 and c_1 denote the unit length resistance and capacitance values for die 1. r_2 and c_2 are similarly defined for die 2. The capacitance and resistance of a through via connecting the two die are denoted C_v and R_v. Let $p_0, p_1, p_2.....p_m$ be the pins of a net, with p_0 being the driver pin. The Elmore delay at a sink pin p is defined as

$$d(p) = \sum_{k \in p_0 \rightarrow p} r_k C_k$$

where r_k is the resistance along the $p_0 \rightarrow k$ path, and C_k is the downstream capacitance as seen from node k. Given a pin p and an edge $e \in T$, the *connection point* is defined as the point on e to which p is connected. To determine the location of the connection point, we need to consider the following two cases:

- Case 1: T consists only of the driver pin p_0. In this case, if p lies on the same die as p_0, $d(p)$ is computed based on the shortest rectilinear route between p_0 and p. If p lies on a different die, $d(p)$ depends on the location of the through via as well as the amount of interconnect assigned in both die.

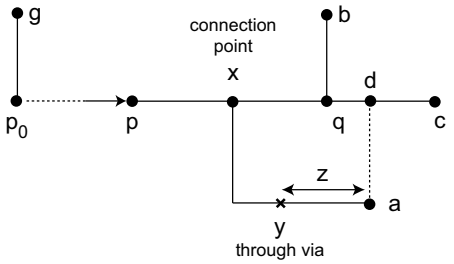

Figure 4: Illustration of how pin a connects to $e(p,c) \in T$ that lies in a different die. x is the location of connection point on $e(p,c)$. y is the location of the through via inserted on $e(x,a)$. $e(q,b)$ is another branch in T. g is another sink that is not a part of the subtree rooted at p. d is the shortest distance point on $e(p,c)$ from a, and z is the distance between the through via and a. The Elmore delay of $T \cup a$ is a function of both x and y. $e(y,a)$ is routed in the bottom die, whereall as all othe edges are routed in the top die.

- Case 2: p connects to an edge $e \in T$. In this case, if p and e lie on the same die, it was shown in [9] that p will connect to either (i) the source node of e or (ii) the point on e that is of the shortest distance to p. If p and e lie on different die, the delay of *all* nodes in T including p depend on the location of connection point as well as the through via.

The following Section 4.3 provides details on how to compute the location of connection points and through vias in 3D case.

4.3 Connection Point and Via Location

The connection point computation for 2D case has been presented in [9], where the Elmore delay change on an entire tree caused by adding a new pin to the tree is a function of a single variable x, the location of connection point. We extend this work by introducing a second variable y that represents the location of through via. We then optimize the two-variable delay function and determine the optimal location of connection point ($= x$) and through via ($= y$) for 3D case. Refereing to Figure 4, $e(p,c)$ and $e(q,b)$ are edges on T. p is the parent node of $e(p,c)$, and q is the parent node of $e(q,b)$. a is the new pin that needs to connect to $e(p,c)$. Edge $e(p,c)$ lies on die 1 with interconnect parasitics r_1 and c_1, whereas a lies on die 2 with interconnect parasitics r_2 and c_2. d is the point on $e(p,c)$ that is of the shortest distance to a. x is the connection point, and y is the location of through via.

Our first goal is to derive Elmore delay equations that are the functions of x and y. In what follows, we let x denote the distance between node p and node x for the simplicity of the discussion. q, a, b, c, and d are used similarly. y is the distance between x and y, and z is the distance between y and a. Let T_b denote the subtree rooted at node b. In order to compute the Elmore delay change on all sink pins in T caused by adding a to T, we consider the following four cases: (i) delay at the node to be added (= node a), (ii) delay at the subtree located after the connection point (= node c), (iii) delay at the subtree that could be located either before or after the connection point (= node b), (iv) delay the nodes not in T_p.

- Case 1: delay at node a. In this case, $d(a)$ is a sum of four functions. f_1 is the delay from p_0 to p. f_2 is the delay from p to x without considering $e(q,b)$ and T_b. f_3 is the delay from p to x when considering $e(q,b)$ and T_b. f_4 is the delay from x to a. Thus we have:

$$d(a) = f_1 + f_2 + f_3 + f_4$$

where

$$f_1 = K_0 + K_1\{c_1 y + C_v + c_2 z + c_1 c + C_c + C_b + c_1(b-q)\}$$

$$f_2 = r_1 x \left\{ c_1 \frac{x}{2} + c_1 y + C_v + c_2 z + c_1(c-x) + C_c \right\}$$

$$f_3 = \begin{cases} r_1 x(b - q + C_b), & \text{if } x \le q \\ r_1 q(b - q + C_b), & \text{if } x \ge q \end{cases}$$

$$f_4 = r_1 y \left(c_1 \frac{y}{2} + C_v + c_2 z \right) + R_v \left(\frac{C_v}{2} + c_2 z \right) + r_2 c_2 \frac{z^2}{2}$$

where $z = a-(x+y)$, K_0 is the sum of resitance/cpacitance products along $p_0 \rightarrow p$ path, K_1 is the sum of resistance along $p_0 \rightarrow p$ path, and C_i is the capacitance of the sub-tree rooted at i^{th} node.

- Case 2: new delay at node c. The delay is given by

$$d(c) = f_1 + f_2 + f_3' + f_4'$$

where

$$f_3' = r_1 q(b - q + C_b)$$

$$f_4' = r_1(c-x)\left\{ \frac{c_1(c-x)}{2} + C_c \right\}$$

- Case 3: new delay at node b. The delay is given by

$$d(b) = f_1 + f_2'' + f_3''$$

where

$$f_2'' = \begin{cases} r_1 x(c_1 y + C_v + c_2 z), & \text{if } x \le q \\ r_1 q(c_1 y + C_v + c_2 z), & \text{if } x \ge q \end{cases}$$

$$f_3'' = r_1 q \left\{ c_1 \frac{q}{2} + c_1(b-q) + C_b + C_c \right\}$$

- Case 4: For all other nodes not in T_p (node g in Figure 4), the added delay is a function of the added capacitance, which is linear in terms of x and y and given by

$$\Delta C = c_1(x+y) + C_v + c_2 z$$

4.4 Optimization of Delay Equations

In general, for a quadratic function of two variables $f(x,y)$, the maximum or the minimum of the function depends upon the values of $\frac{\partial^2 F}{\partial x^2}$ and the determinant of the Hessian matrix H_1:

$$\begin{vmatrix} \frac{\partial^2 F}{\partial x^2} & \frac{\partial^2 F}{\partial x \partial y} \\ \frac{\partial^2 F}{\partial x \partial y} & \frac{\partial^2 F}{\partial y^2} \end{vmatrix}$$

where F is the delay function under consideration. The above values for a quadratic function of two variables are always constant. For our case we have $0 \le x \le d$ and $0 \le y \le a$ and thus consider the following cases:

- Case 1: If $\frac{\partial^2 F}{\partial x^2} \le 0$ and $H_1 \ge 0$, the minimum can be found at the boundary points, i.e., $x = 0$ or $x = d$ and $y = 0$ or $y = a$. Thus we have four points to look for the minimum.

- Case 2: If $\frac{\partial^2 F}{\partial x^2} \le 0$ $\frac{\partial^2 F}{\partial y^2} \le 0$ and $H_1 = 0$, we have a concave function, and the minimum lies on the boundary points.

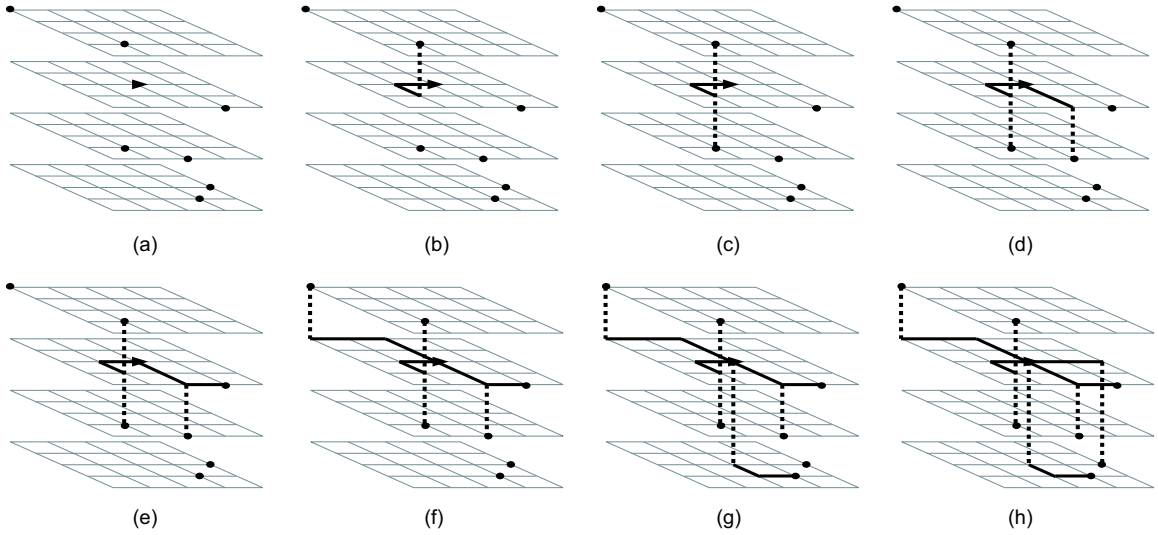

Figure 5: Illustration of 3D Steiner tree construction.

- Case 3: If $\frac{\partial^2 F}{\partial x^2} = 0$ $\frac{\partial^2 F}{\partial y^2} = 0$ and $H_1 = 0$, then $f(x, y)$ is a linear function of x and y, and the minimum lies at the boundary points.

- Case 4: If $H_1 < 0$, the critical point found is a saddle point. and the minimum lies at the boundary, although a different set of boundary points may need to be chosen. The set of boundary points may be found by setting $x = 0$ or $x = d$ and minimizing $f(x, y)$ as a function of y or setting $y = 0$ or $y = a$ and minimizing $f(x, y)$ as a function of x.

We show that the Elmore delay at each sink node in T can be optimized by considering any of the 4 cases shown above. Thus, there is a fixed number of points (x, y) for which the Elmore delay can be minimized. Assuming $a_0 = r_2/r_1$ and $b_0 = c_1/c_2$, the optimization of two-variable delay functions allow the computation of x (= connection point) and y (= through via location) as follows:

- For $d(a)$ we have $\frac{\partial^2 F}{\partial x^2} = r_1 c_2(a_0 - b_0 - 2)$, $\frac{\partial^2 F}{\partial y^2} = r_1 c_2(a_0 + b_0 - 2)$, and $H_1 = -(r_1 c_2)^2\{(a_0 + b_0 - 2)2b_0\}$. Thus, we see that when $H_1 = 0$, $\frac{\partial^2 F}{\partial x^2} \leq 0$ and $\frac{\partial^2 F}{\partial y^2} = 0$, the optimal delay is found at points according to the Case 2. If $H_1 < 0$, optimal delay is found at points in according to the Case 4. If $H_1 > 0$, we have $\frac{\partial^2 F}{\partial x^2} \leq 0$, so the optimal delay is found at points according to the Case 1.

- For $d(b)$ we need to evaluate two cases: (i) when $x \geq b$, we have $\frac{\partial^2 F}{\partial x^2} = 0$, $\frac{\partial^2 F}{\partial y^2} = 0$, and $H_1 = 0$. Thus, the optimal delay is found at points according to the Case 3. (ii) when $x \leq b$, we have $\frac{\partial^2 F}{\partial x^2} = -2r_1 c_2$, $\frac{\partial^2 F}{\partial y^2} = 0$, and $H_1 = -(r_1 c_2)^2(b_0 - 1)^2$. Thus, if $H_1 = 0$, the optimal delay is found at points according to the Case 2. Otherwise, they are found at points according to the Case 4.

- For $d(c)$ we have $\frac{\partial^2 F}{\partial x^2} = -2r_1 c_2$, $\frac{\partial^2 F}{\partial y^2} = 0$, and $H_1 = -(r_1 c_2)^2(b_0 - 1)^2$. If $H_1 = 0$, the optimal delay is found at points according to the Case 2. Otherwise, they are found at points according to the Case 4.

- For all other nodes not in T_p, we have $\frac{\partial^2 F}{\partial x^2} = 0$, $\frac{\partial^2 F}{\partial y^2} = 0$, and $H_1 = 0$ since delay is a linear function of x and y. Thus, the optimal delay is found at points according to the Case 3.

Since a_0 and b_0 values are dependent on the interconnect parameters at each die, we can see that the number of points (x, y) to which a pin can connect to an edge in 3D case is a fixed constant. At each stage of the 3D Steiner tree generation, out goal is to choose one of these points which gives us the minimum max-delay among all sink nodes of the existing tree. Figure 5 shows an ilustration of 3D Steiner tree construction using our algorithm.

5. EXPERIMENTAL RESULTS

We implemented our algorithms in C++/STL and ran our experiments on Linux PC running at 2GHz. We tested our algorithms with two sets of benchmark: ISCAS89 and ITC99. We do not use ISPD98 circuits due to the absence of signal direction information. Our experiment is based on 4-die stacked IC, where the top two and bottom two dies are bonded in face-to-face and the middle two in back-to-back. We assume all 4 dies have the same unit-length resistance and capacitance values, but our algorithm is applicable for heterogeneous cases. We use the interconnect parameters from [4]: $r_u = 86\Omega/mm$ and $c_u = 396fF/mm$ for unit-length interconnect for all four dies. $R_v = 53\Omega/mm$ and $C_v = 280fF/mm$ for F2F through vias, and $R_v = 100\Omega/mm$ and $C_v = 650fF/mm$ for B2B through vias.

Table 1 shows our layer partitioning results. The first cutline is B2B (constrained mincut) while the subsequent two cutlines are F2F (constrained maxcut). The F2F and B2B via constraints are calculated based on the size of each circuit. We first report the cutsize of the 1st, 2nd, and 3rd bipartitioning. From the comparison between (i) B2B via limit vs 1st cut, and (ii) F2F via limit vs 2nd/3rd cuts, we observe that our cutsizes indeed satisfied the cutsize constraints. Next, we report the timing-weighted cutsize improvement from the point we hit the cutsize target until the current bipartitioning process is completed. Our partitioning change its target from "pure" cutsize to "timing-weighted" cutsize when the cutsize first hits the target. Then the weighted cutsize is either maximized or minimized depending on the bonding style. The pure cutsize should stay within a limited range from the target during

Table 1: Layer partitioning results.

	via limit		cutsize			timing imp (%)		
ckt	B2B	F2F	1st	2nd	3rd	1st	2nd	3rd
b14_opt	520	2400	515	2403	2335	9.8	4.6	3.5
b15_opt	750	3400	742	3375	3101	11.2	4.6	4.3
s9234	530	2500	526	2502	2249	13.4	2.5	4.5
s15850	980	4400	936	4391	4001	16	3.3	3.2
s13207	820	3700	785	3679	3313	12	3.1	3.7
b20_opt	1150	5600	1125	5593	4497	15	3.9	4.1
b17_opt	2400	11000	2339	10983	9875	11.1	4.0	3.9
s38417	2200	11000	2166	10223	9800	10.1	2.8	2.8

Table 2: 3D Steiner tree construction results for 3D maze router and our 3D Elmore router. Congestion is not considered.

	3D Maze Routing			3D Elmore Routing		
ckt	wire	delay	via	wire	delay	via
b14_opt	5035	18.2	8331	6723	2.5	9221
b15_opt	6508	24.9	12027	7577	3.1	12153
s9234	4091	6.8	6832	4706	2.2	6896
s15850	7719	17.8	12151	8792	3.4	12231
s13207	6523	8.3	10119	7445	2.2	10215
b20_opt	11273	22.4	19429	14728	3.3	19493
RATIO	1.00	1.00	1.00	1.20	0.17	1.01
TIME	515 sec			216 sec		

this process. We observe from the Table 1 that this scheme indeed improves the weighted cutsize further. Thus, we conclude that we should not terminate the partitioning process once the target cutsize is achieved since the subsequent cell moves can optimize other objectives such as timing.

For our 3D Steiner tree generation algorithm, we compare our results with a wirelength-driven 3D maze router. Our 3D maze router each time finds the shortest path from the source node to the nearest sink. Then all nodes along the path become new source nodes. We report the total wirelength, number of vias, and the maximum delay among all sinks for each circuit. Table 2 shows a comparison when congestion is not considered, i.e., the net/via capacity constraints are not imposed. We observe that our 3D Elmore router achieves a significant delay improvement over 3D maze router. The total via used is about the same in both cases. The wirelength, however, has increased noticeably. This is expected since the maze router tends to form chain-like routes, whereas the Elmore router prefers star-shaped topologies. An illustration is show in Figure 6. Total runtime for all circuits are reported as well. We note that 3D maze router is slightly slower than 3D Elmore router.

We consider the congestion issue in Table 3, where we compare (i) congestion-driven 3D maze router and (ii) a hybrid 3D Elmore routing plus 3D maze rip-up-and-reroute. The rationale behind the hybrid approach is to first perform 3D Elmore routing for timing critical nets and then reroute non-timing critical nets for congestion reduction. We observe that the maze-based rerouting degrades the delay as well as via results. However, all the net/via capacity constraints are satisfied. The final delay result is still better for the hybrid approach.

6. CONCLUSIONS

This paper studied two new problems that are important to the through via management for 3D stacked IC technology: bonding-aware layer partitioning and 3D Steiner tree construction. We formulated the constrained min/max partitioning problem to handle netlist partitioning into multiple die while considering various types of bonding styles available in 3D ICs. We for the first time formulated and solved the 3D Steiner routing problem. Our algorithm is based on a constructive method, where the Steiner tree is grown by connecting a new pin to the existing tree. We derived two-variable delay equations and optimized them to compute the Elmore delay optimal location of the through vias. Our future work will address process, thermal, and voltage variation issues during the Steiner tree construction.

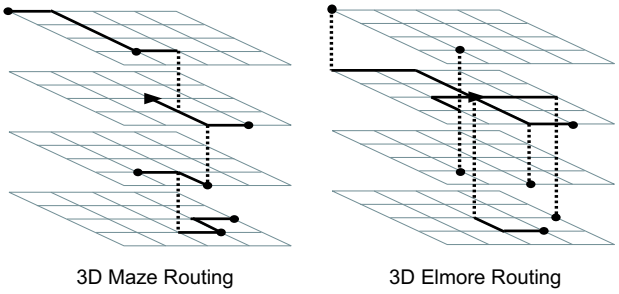

Figure 6: Comparison between 3D maze vs 3D Elmore routing results. The wirelength of 3D

Table 3: Congestion-aware 3D Steiner tree construction results for 3D maze router and our 3D Elmore router.

	Cg-aware 3D Maze			3D Elmore + Maze		
ckt	wire	delay	via	wire	delay	via
b14_opt	5035	18.7	8331	6349	7.8	9552
b15_opt	7486	25.3	11462	8394	6.1	11462
s9234	4094	6.9	6832	4686	3.2	6910
s15850	7719	17.8	12154	8685	3.4	12423
s13207	6523	8.1	10201	7148	2.7	10231
b20_opt	11387	22.4	19439	15001	10.9	21620
RATIO	1.00	1.00	1.00	1.17	0.34	1.06
TIME	518 sec			447 sec		

7. REFERENCES

[1] A. Fan, A. Rahman, and R. Reif. Copper wafer bonding. *Electrochemical Solid-State Letter*, 1999.

[2] S. B. Horn. Vertically Integrated Sensor Arrays VISA. In *Defense and Security Symposium*, 2004.

[3] M. Umemoto, K. Tanida, Y. Nemoto, M. Hoshino, K. Kojima, Y. Shirai, and Kenji Takahashi. High-Performance Vertical Interconnection for high-density 3D Chip Stacking Package. In *IEEE Electronic Components and Technology Conf.*, 2004.

[4] V.F. Pavlidis and E.G. Friedman. Interconnect Delay Minimization through Interlayer Via Placement in 3-D ICs. In *Proc. Great Lakes Symposum on VLSI*, 2005.

[5] C. Sechen and A. Sangiovanni-Vincentelli. The TimberWolf Placement and Routing Packag. In *IEEE Journal of Solid-State Circuits. vol 20*, 1985.

[6] J. Cong, J. Wei, and Y. Zhang. A Thermal-Driven Floorplanning Algorithm for 3D ICs. In *Proc. IEEE Int. Conf. on Computer-Aided Design*, 2004.

[7] Brent Goplen and Sachin Sapatnekar. Efficient Thermal Placement of Standard Cells in 3D ICs using a Force Directed Approach. In *Proc. IEEE Int. Conf. on Computer-Aided Design*, 2003.

[8] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel Hypergraph Partitioning : Application in VLSI Domain. In *Proc. ACM Design Automation Conf.*, pages 526–529, 1997.

[9] K. Boese, A. Kahng, B. McCoyy, and G. Robins. Near-Optimal Critical Sink Routing Tree Constructions. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 1995.