Assembling 2D Blocks into 3D Chips

Johann Knechtel[†‡], Igor L. Markov[†] and Jens Lienig[‡]
†University of Michigan, EECS Department, Ann Arbor USA
‡Dresden University of Technology, EE Department, Dresden Germany
johann.knechtel@ifte.de, imarkov@eecs.umich.edu, jens.lienig@ifte.de

ABSTRACT

Three-dimensional ICs promise to significantly extend the scale of system integration and facilitate new-generation electronics. However, progress in commercial 3D ICs has been slow. In addition to technology-related difficulties, industry experts cite the lack of a commercial 3D EDA tool-chain and design standards, high risk associated with a new technology, and high cost of transition from 2D to 3D ICs. To streamline the transition, we explore design styles that reuse existing 2D Intellectual Property (IP) blocks in 3D ICs. Currently, these design styles severely limit the placement of Through-Silicon Vias (TSVs) and constrain the reuse of existing 2D IP blocks in 3D ICs. To overcome this problem, we develop a methodology for using TSV islands and novel techniques for clustering nets to connect 2D IP blocks through TSV islands. Our empirical validation demonstrates 3D integration of traditional 2D circuit blocks without modifying their layout for this context.

Categories and Subject Descriptors

B.7.2 [**Integrated Circuits**]: Design Aids — *Layout*

General Terms

Algorithms, Design

Keywords

3D integration, IP blocks, design styles, TSV islands

1. INTRODUCTION

Modern System-on-Chip (SoC) design faces numerous challenges, as steadily increasing demands on functionality and performance push against the limits of semiconductor manufacturing and EDA tools. Recent process-technology advances promise shorter interconnect and greater device density by means of three-dimensional integration — stacking multiple dies and implementing vertical interconnections with *Through-Silicon Vias (TSVs)*. Such 3D ICs can significantly extend the scale of system integration and facilitate new-generation electronics. While progress in commercial

3D ICs has been slow, memory-on-logic stacking has reached the market.[1] Several major design and manufacturability issues with 3D ICs currently remain unsolved [13], and a definitive commitment to 3D integration typically requires favorable cost considerations, as well as the availability of 3D EDA tools, industry standards and design methodologies. Industry experts are additionally concerned about the high risk associated with such a new technology, and potentially-prohibitive cost of transition from 2D to 3D ICs. To streamline this transition, we propose to focus on design styles that reuse existing 2D Intellectual Property (IP) blocks. As is well-known, modern chip designs are dominated by 2D IP blocks, proven in applications and considered reliable. Thus, we advocate 3D integration of legacy 2D IP blocks to circumvent many of the obstacles that currently impede wide adoption of 3D ICs.

In this paper, we make the following contributions. **First**, we describe and compare several possible design styles for 3D integration of 2D blocks, in particular the *Legacy 2D (L2D)* style which *integrates existing IP blocks* not designed for 3D integration. **Second**, we introduce a new design style for 3D integration of 2D blocks, called L2Di, where TSVs are clustered into TSV islands to reduce area overhead and provide post-silicon self-repair (similar to that in DRAM). **Third**, we propose novel algorithms and methodologies for net clustering, TSV-island insertion, deadspace alignment, and related tasks. The overall approach promises faster industry acceptance of 3D integration of legacy 2D IP blocks. **Fourth**, we empirically validate our methodology, demonstrating 3D integration of legacy 2D IP blocks without modifying their layout.

The remainder of this paper is structured as follows. In Section 2 we review important TSV characteristics and resulting integration challenges. In Section 3 we contrast possible design styles for 3D integration and discuss clustering of TSVs into TSV islands. We provide the problem formulation for the L2Di-style 3D integration in Section 4. Next, we describe our methodology in Section 5. In Section 6 we provide an empirical validation and in Section 7 we give our conclusions.

2. BACKGROUND

Since adjacent dies are connected by TSVs, TSVs are critical to 3D integration. TSVs can be manufactured in two ways: *via-first* and *via-last*. Via-first TSVs are 1-5μm in diameter and fabricated before the final metallization process; via-last TSVs are 5-20μm and fabricated after final metallization [15]. Furthermore, manufacturability demands *landing pads* and *keep-out zones* [31] which further increase TSV area footprint. At the 45nm technology node, the area

---

[1]For example, the recently released *Apple A4* SoC contains an ARM core die and two memory dies. However, the vertical interconnections are realized using wire-bonding, not TSVs.
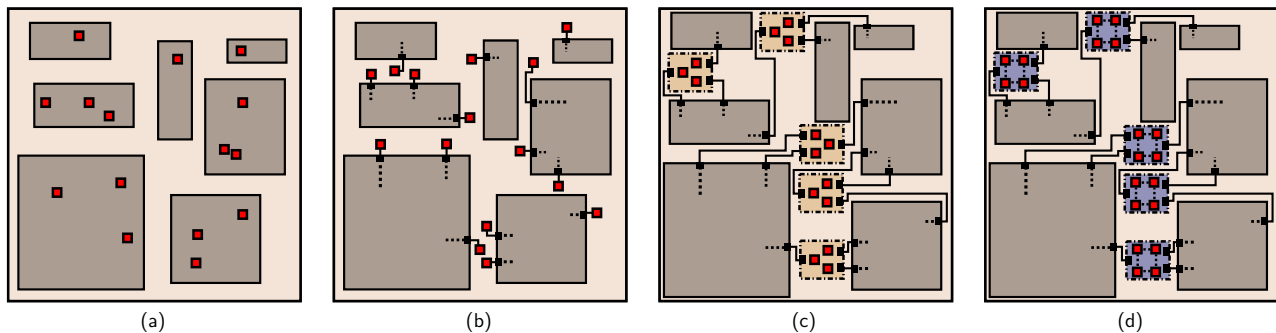
Figure 1: TSV positioning in design styles for 3D integration. (a) Gate-level and Redesigned 2D (R2D) styles place TSVs (small boxes) within the block footprint. (b) Legacy 2D (L2D) style places scattered TSVs between blocks, (c) L2D style with TSV islands (L2Di) groups TSV to blocks. (d) TSV islands can incorporate scan chains for TSV test and multiplex spare TSVs for redundancy.

footprint of a $10\mu m \times 10\mu m$ TSV is comparable to that of about 50 gates [16]. Hence, 10,000 TSVs can displace half a million gates. Previous work in physical design often neglects design constraints and overhead associated with TSVs, especially their area footprint [3, 11, 21–23, 28, 32]. Some studies explicitly consider thermal TSV insertion but not signal TSVs [20, 22, 23, 29]. Other studies incorporate signal and power TSVs in their flow, but sometimes ignore footprints of signal TSVs [18, 19].

Wirelength impact of TSVs. While the usage of TSVs is generally expected to reduce total wirelength, Kim et al. [16] observe that wirelength reduction varies depending on the number of TSVs and their characteristics. They show that TSV insertion in general and the impact of TSV footprint area in particular may increase silicon area and/or routing congestion, thereby making wires longer. Consequently, Kim et al. propose a new wirelength distribution model to estimate wirelength reduction while considering the impact of TSVs on wirelength. Case studies in [16] show that excessive usage of TSVs can undermine their potential advantages, and that this trade-off is controlled by the *granularity* of inter-die partitioning. The wirelength typically decreases for moderate (blocks with 20-100 modules) and coarse (block-level partitioning) granularities, but increases for fine (gate-level partitioning) granularities.

Another study by Kim et al. [15] suggests limiting the number of TSVs — otherwise, wirelength reduction can be undermined. Their study also reveals that using four to six dies offers most benefit for wirelength reduction. However, die stacking also raises mechanical issues, such as material stress and accurate TSV alignment [24, 31], and may increase the overhead of test structures [17], exacerbate thermal problems [2], and increase the impact of intra-die variation [7]. Due to these major complications, practical 3D integration begins with only two active layers, as illustrated by a recently taped-out memory-on-processor design [9].

TSVs as layout obstacles. Via-first TSVs occupy the device layer, resulting in placement obstacles, while via-last TSVs occupy both the device and metal layers, resulting in placement and routing obstacles [16]. Hence, TSVs must be accounted for during floorplanning and/or placement. A study by Kim et al. [15] compares placing TSVs on a grid (*regular placement*) to placing scattered TSVs (*irregular placement*). The study reveals that irregular placement performs better in terms of wirelength reduction and design runtime. Since the TSVs are placed near the blocks they are connected to, there is no need for a separate TSV pin assignment process. However, regular placement helps manufacturing reliable TSVs [9, 10]. This also applies to TSV islands.

3. 3D IC DESIGN STYLES

3D integration originated with package-level integration, which connects multiple 2D chips through bonding pads, as illustrated by the quad-core variant of the *Intel Core 2* processor (two cores per die, two dies in one package). Finer granularity of 3D integration is enabled by connecting dies with TSVs, which results in 3D ICs [5]. In this section, we first discuss gate-level and block-level integration styles for 3D ICs. Gate-level integration faces multiple challenges and currently appears less practical than block-level integration. Second, we introduce a promising approach to 3D block-level integration, the Legacy 2D (L2D) style. It vertically integrates existing 2D blocks that were not originally designed for 3D integration. Third, we explain how 2D blocks are connected through TSVs.

3.1 Gate-Level Integration

One approach to 3D integration is to partition standard cells between multiple dies in a 3D assembly and use TSVs in routes that connect cells spread among active layers. This integration style promises significant wirelength reduction and great flexibility [2]. Its adverse effects include the massive number of necessary TSVs for random logic, as discussed in Section 2. The study by Kim et al. [16] reveals that partitioning gates between multiple dies may undermine wirelength reduction unless circuit modules of certain minimal size are preserved. In addition, partitioning a design block across multiple dies means it cannot be fully tested before die stacking. Moreover, after die stacking (post-bond testing), a single failed die can render several good dies unusable, thus undermining yield. Fine-grain partitioning between active layers also amplifies the impact of process variation, especially inter-die variation, on critical paths [7]. Monte Carlo SPICE simulations in [7] show that a 3D design is less likely to meet timing constraints than a comparable 2D design. Furthermore, gate-level 3D integration requires redesign of all blocks since existing IP blocks and EDA tools do not provision for 3D integration. However, 3D place-and-route tools are not yet available on the market, and IP providers have heavily invested in legacy 2D IP blocks.

3.2 Block-Level Integration

Design blocks subsume most of the netlist connectivity and are linked by a smaller number of global wires. Therefore, block-level integration reduces TSV overhead. The assignment of entire blocks to separate dies can be performed in two ways (Figure 1).

- *Redesigned 2D (R2D) style*: 2D blocks designed for 3D integration (TSVs included within the block footprints)
- *Legacy 2D (L2D) style*: 2D blocks not designed for 3D integration (TSVs placed between blocks)
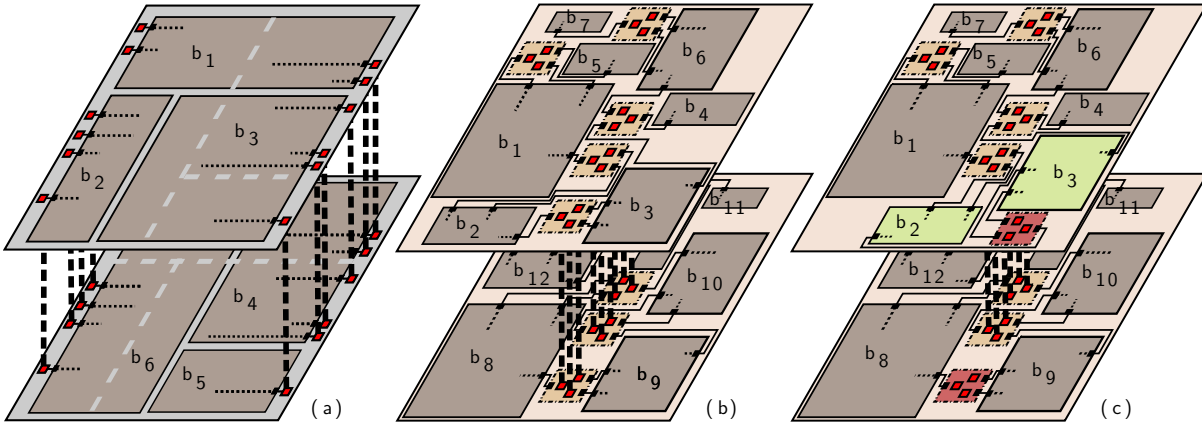
Figure 2: Deadspace alignment. Gray lines indicate (deadspace) channels and their mirror images on the adjacent die (gray dashed lines). (a) Floorplans that are small and/or regular usually contain little deadspace. Given insufficient aligned deaspace, vertical interconects (TSVs, wire-bonding) are placed at the boundary of the chip. (b) Floorplans with numerous blocks usually contain more deadspace and thus allow TSV-island insertion. Each TSV (island) requires adequate deadspace at two layers, with vertical alignment. (c) Poor alignment results in deadspace unusable for TSV insertion. A design example is shown in Figure 8.

These block-level integration styles promise the best trade-off between necessary TSV usage and wirelength reduction (Section 2). Several other important benefits of block-level integration are described next. Existing IP blocks are already equipped with Design for Testability (DFT) structures and can be tested before individual dies are stacked (pre-bond testing) [17].[2] With block-level integration, critical paths are mostly located within 2D blocks — they do not traverse multiple active layers, which limits the impact of TSV and inter-die variation on manufacturing yield. In [4], the authors propose optimal matching of *slow* and *fast* dies, based on accurate delay models with process variations considered. However, we note that this approach assumes that dies can be delay-tested before 3D stacking — a strong argument for block-level 3D integration.

Another aspect of block-level integration styles deals with design effort. The R2D style implies redesigning existing IP blocks, despite their successful track record in applications. This may require new EDA tools for physical design and verification, increasing risks of design failures and being late to market. Therefore, one hopes to avoid redesigning the broad spectrum of available functionality. It is more convenient to use legacy 2D IP blocks and to place the mandatory TSVs in the deadspace between the blocks, as provisioned by the L2D style. An extreme form of design IP reuse possible with the L2D style is *block-level mask reuse* with changes only required for global routes at high metal layers — TSVs placed in deadspace do not modify silicon layers of the blocks.

3.3 Connecting 2D Blocks by TSVs

The L2D style admits both scattered TSVs (Figure 1(b)) and TSV clusters (Figures 1(c,d)) In both cases, IP blocks are connected to TSVs with dedicated routes. TSV clusters require longer connections to block pins, but improve manufacturability by increasing exposure quality during optical lithography [10].

Clustering TSVs into TSV islands is helpful for multiple reasons. First, TSVs introduce stress in surrounding silicon which affects nearby transistors [31], but TSV islands do not need to include logic gates. The layout of TSV islands can be optimized in advance by experienced engineers. Second, clustering TSVs facilitates TSV-redundancy architectures [10, 24], where failed TSVs are shifted

within a chain structure or dynamically rerouted to spare TSVs. For example, Figure 1(d) illustrates islands with four TSVs, one of wich is spare. For the L2D style with TSV islands (L2Di), TSV islands can be placed in the deadspace between blocks.

Deadspace alignment. In some chip floorplans, the deadspace between blocks is too small to accommodate the necessary number of TSVs. In such cases, vertical interconnects can be realized by wire-bonding pads at the chip periphery or by injecting additional deadspace at the cost of a larger die footprint (Figure 2(a)). For larger designs with many diverse blocks, it is easier to find deadspace for TSV-island insertion. However, given a region of deadspace on a particular die, a TSV island also requires another region of deadspace on a neighboring die, with adequate alignment between the two regions.[3] Figure 2(b) illustrates a feasible TSV-island placement thanks to proper deadspace alignment while Figure 2(c) illustrates an infeasible TSV-island placement — blocks may need to be shifted to create sufficient aligned deadspace. In some cases, die area may need to be increased to accommodate TSV islands.

TSV placement and routing. TSV islands must be placed in aligned deadspace to facilitate short inter-layer routes that connect block pins through TSVs. A given net may be routed through one or several TSVs (islands). 2D routes may use high metal layers and/or channels between 2D blocks. The work in [30] dedicates an entire chip level to interconnect fabric.

4. PROBLEM FORMULATION

The L2Di style for 3D integration assumes the following input.

(i) **Active layers**, denoted as set $\mathcal{L}$. Each layer $l \in \mathcal{L}$ has dimensions (h_l, w_l) such that every block assigned to l can fit in the outline without incurring overlap. Note that the outlines may differ and thus allow layers to be shifted to improve deadspace alignment.

(ii) **Rectangular IP blocks**, denoted as set $\mathcal{B}$. Each block $b \in \mathcal{B}$ has dimensions (h_b, w_b) and pins, denoted as set $\mathcal{P}^b$. Each pin $p \in \mathcal{P}^b$ of block b is defined by its offset (δ_p^x, δ_p^y) with respect to the block's geometric center (origin).

---

[2]Test pins can be provisioned on each die and multiplexed/shared with other pins for pre- and post-bond testing [14].

[3]Alignment can be achieved by shifting blocks within each die, and also by displacing the dies, if the 3D chip package allows.
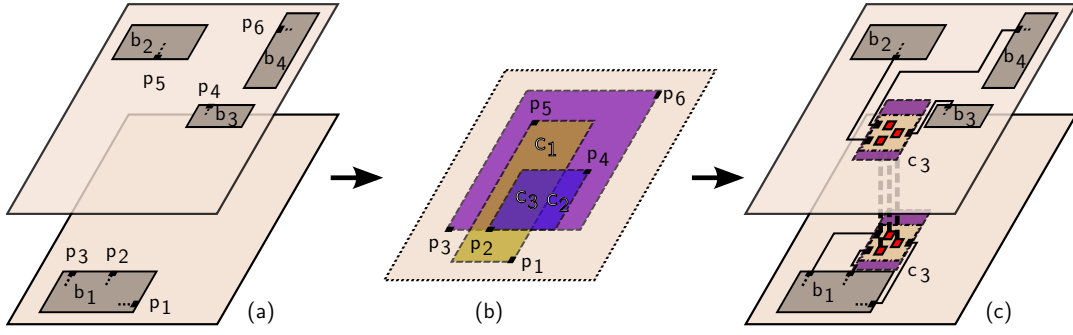
Figure 3: Net clustering and TSV-island insertion. (a) Inter-layer nets $n_1 = \{p_1, p_5\}$, $n_2 = \{p_2, p_4\}$, and $n_3 = \{p_3, p_6\}$ need to be connected through TSVs. (b) Pins are mapped to a virtual die and net bounding boxes are created. Intersections of bounding boxes mark cluster regions c_1, c_2, and c_3. The cluster region c_3 represents desired locations for a TSV island, which would facilitate shortest routes for all nets. (c) This particular cluster region is not obstructed by blocks and provides sufficient area, thus allowing TSV-island insertion without block shifting.

(iii) **Netlist**, denoted as set $\mathcal{N}$. A net $n \in \mathcal{N}$ describes a connection between two or more pins.

(iv) **TSV-island types**, denoted as set $\mathcal{T}$. Each type $t \in \mathcal{T}$ has dimensions (h_t, w_t) and capacity κ_t. Since pre-designed TSV-island types may incorporate spare TSVs, κ_t defines the number of nets that can be routed through t. While a simple formulation may deal with one type only, we believe that using pre-designed types of different shapes is essential to facilitate TSV-island insertion.

(v) **3D floorplan**, denoted as set $\mathcal{F}$. Each block b is assigned a location (x_b, y_b, l_b) such that no blocks overlap. (x_b, y_b) denotes the coordinate of the block's origin, l_b denotes the assigned layer.

3D integration with the L2Di style seeks to cluster inter-layer nets into TSV islands, and to insert TSV islands into aligned deadspace around floorplan blocks. If TSV-island insertion is impossible due to lack of aligned deadspace, blocks can be shifted from their initial locations, but their relative positions must be preserved. Figures 2(b,c) illustrate such block shifting. If TSV-island insertion is still impossible, additional deadspace can be inserted.

5. OUR METHODOLOGY

To connect blocks on different dies following the L2Di style, we need the locations of TSV islands. However, these locations must account for routes, so as to avoid unnecessary detours. In order to solve this chicken-and-egg problem, our techniques (i) clusters nets to estimate global routing demand, and (ii) uses these clusters to iteratively insert TSV islands. Details of our techniques are discussed in the following subsections, the overall flow is illustrated in Figure 4. For clarity of exposition, this section illustrates 3D integration in the case of only two dies. However, our techniques can be extended to more than two dies as well. In the following discussion, we refer to inter-layer nets as just nets.

Global iterations. Our clustering algorithm relies on a uniform grid. Grid-tile sizes influence per-tile net count. For example, quartering grid-tile size in Figure 5(b) would decrease the maximum per-tile net count from four to two. Having fewer nets per tile reduces the cluster size, increasing chances of TSV-island insertion. Therefore, we wrap our clustering and TSV-island insertion algorithm into an outer loop, which iteratively decreases grid-tile size from an upper bound f_{max} to a lower bound f_{min} (Table 1).
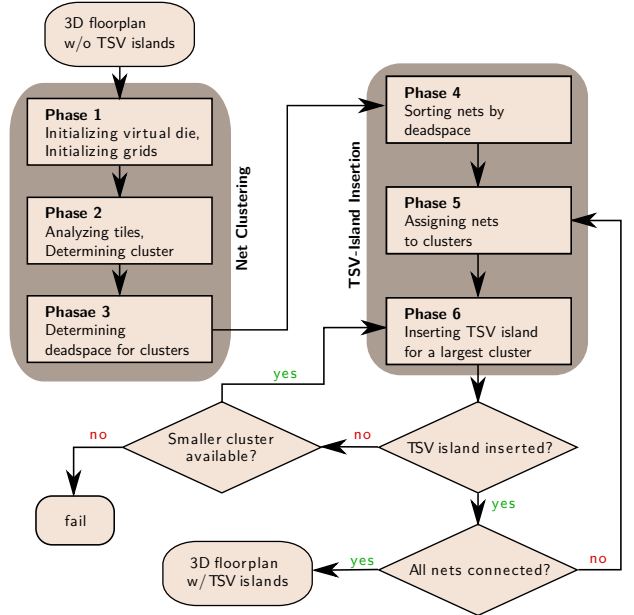
5.1 Net Clustering

The rationale for clustering nets is that placing TSV islands within net bounding boxes facilitates shortest-path connections. Also, as-

signing nets to clusters helps to select the type and capacity of each TSV island. Figure 3(c) illustrates a cluster of three nets routed through a TSV island.

To formalize the clustering process, we consider a *virtual die* — the minimum rectangle containing projections of active-layer outlines. The pins of each net are projected onto the virtual die (Figure 3(b)). Intersections of net bounding boxes in the virtual die suggest possible locations of TSV islands (Figure 3(c)).

Relevant results from graph theory. Imai and Asano [12] conducted a study on intersections of axis-aligned rectangles in the plane. First, Imai and Asano proved that n axis-aligned rectangles (e.g., bounding boxes) have a single non-empty n-way intersection *iff* each pair of these rectangles overlap. Thus, rather than check all subsets of overlapping bounding boxes, we may search

Figure 4: Flow of net clustering and TSV-island insertion. Our techniques first localize global routing demand while determining possible cluster regions, described by intersections of net bounding boxes. Second, TSV-islands insertion into cluster regions is iteratively attempted, based on dynamic scores. Depending on success of TSV-island insertion, our techniques provide a 3D floorplan with suitable placed TSV islands.
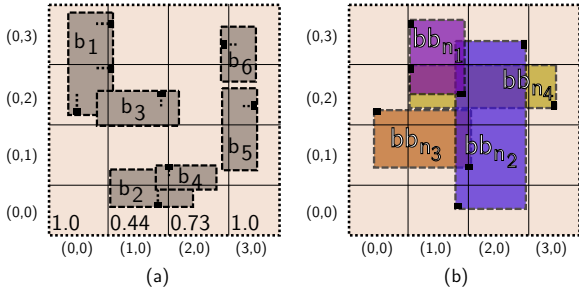
Figure 5: Uniform clustering grid $\mathcal{G}$ on virtual die. (a) Projected blocks from all active layers. We calculate the per-tile ratio of aligned deadspace, illustrated in the last row. (b) Projected net bounding boxes. According to their bounding boxes, we link nets to covered tiles. The intersection of net bounding boxes must be explicitly checked during clustering. In tile $(1, 2)$, e.g., net bounding boxes bb_{n_1}, bb_{n_3} and bb_{n_3}, bb_{n_4} do not overlap pairwise, but all four nets are linked to the tile.

for cliques in a suitably defined *intersection graph*. This graph represents bounding boxes by vertices and connects overlapping boxes by edges. Second, they provided an $\mathcal{O}(n \log n)$-time algorithm for finding the maximum clique in intersection graphs with n vertices, even though this problem is NP-hard for general graphs [26]. In our context, however, a single clique is insufficient — we seek to partition the edges in the intersection graph into a small set of cliques, which is the NP-complete *clique cover* problem [6] (for *interval graphs*, this problem can be solved in polynomial time [8]).

Results by Imai and Asano imply that the largest possible net clusters correspond to maximum cliques in the intersection graph. However, in our context, large cliques may exceed the capacity of the largest available TSV island. Several TSV islands can be combined to implement such a clique, but this increases routing congestion and mechanical stress, and aggravates signal integrity problems [7, 31]. Another problem with using large cliques is that corresponding (small) intersections of net bounding boxes may not include any aligned deadspace, preventing the insertion of a TSV island. On the other hand, a smaller clique would imply fewer bounding boxes and a larger intersection that is more likely to admit TSV-island insertion. Thus, we need to develop our own algorithm for identifying clique covers. Our algorithm is presented next.

Infrastructure used by our clustering algorithm. Initially, all blocks are projected onto the virtual die. In order to identify clusters (cliques) of appropriate size, a *uniform* clustering grid $\mathcal{G}$ is constructed on the virtual die (Figure 5). A clustering grid links each net n to each tile $\Xi \in \mathcal{G}$ covered by its net bounding box bb_n, and thus results in size-limited (appropriate) clusters. Cluster regions must be close to aligned deadspace, otherwise TSV islands cannot be inserted. To calculate the amount of aligned deadspace, a *non-uniform* grid is constructed. Grid lines are drawn through the four edges of each block. Grid tiles not covered by blocks define deadspace. For m blocks overlapping with a particular tile Ξ, deadspace detection runs in $\mathcal{O}(m^2)$ time [29], which is not prohibitively expensive because typically $m < 50$. In the *uniform* clustering grid, tiles with insufficient aligned deadspace ($< \Xi_{min}^d$) are marked as *obstructed*. Key parameters used in our algorithm are defined in Table 1 along with their values.

Our clustering algorithm is illustrated in Figure 6, referenced phases are also illustrated in Figure 4. **In Phase 1**, the virtual die and the grid structures are constructed. Then, each net is linked to each grid tile within the net's projected bounding box (Figure

$\textsc{Cluster\_Nets}(\mathcal{L}, \mathcal{N}, \mathcal{B}, \mathcal{F})$

```
1   // Phase 1: initialize virtual die and clustering grid
2   Initialize_Virtual_Die(L)
3   G = Initialize_Clustering_Grid(N, B, F)
4   // Phase 1: link nets to grid tiles
5   foreach net n ∈ N
6       bb_n = Determine_Bounding_Box(n, B, F)
7       foreach grid tile Ξ ∈ G where Ξ is covered by bb_n
8           append n to Ξ.nets
9   // Phase 2: determine possible clusters
10  foreach grid tile Ξ ∈ G where Ξ.obstructed == FALSE
11      c = Determine_Cluster(Ξ, Ω_min, O_nets, O_link)
12      if c ∉ C
13          insert c into C
14          foreach net n ∈ c.nets
15              n.clustered = TRUE
16      elseif |c.nets| > 0
17          Update_Cluster_Region(c, Ξ)
18  // Phase 2: handle yet unclustered nets
19  progress = TRUE
20  while progress == TRUE
21      Reset(unclustered_nets)
22      foreach net n ∈ N where n.clustered == FALSE
23          append n to unclustered_nets
24      c = Determine_Cluster(unclustered_nets)
25      progress = (c ∉ C)
26      if progress == TRUE
27          insert c into C
28          foreach net n ∈ c.nets
29              n.clustered = TRUE
30  // Phase 3: determine available deadspace
31  foreach cluster c ∈ C
32      foreach grid tile Ξ ∈ G where Ξ is covered by c.bb
33          c.deadspace += Intersection(c.bb, Ξ) × Ξ.deadspace
```

Figure 6: Our clustering algorithm. Input data are described in Section 4. Grid tiles where aligned deadspace is below a threshold Ξ_{min}^d are pre-marked as obstructed.

5(b)). **In Phase 2**, for each unobstructed grid tile the largest cluster is determined — each linked net is considered as long as the resulting intersection of net bounding boxes is non-empty. Moreover, we impose a lower bound Ω_{min} on the overlap area between the intersection and tiles, in order to assure the intersection is covering the unobstructed tile to some minimal degree. We note that intersections in general can overlap more than one tile, depending on the net bounding boxes. An upper bound O_{nets} of nets clustered within each cluster c must not be exceeded. Also, an upper bound O_{link} for clustering each net n to clusters must not be exceeded. Next, we attempt to cluster yet-unclustered nets by relaxing the imposed bounds. Since this step allows one-net clusters, all nets are guaranteed to be clustered afterwards. **In Phase 3**, available aligned deadspace is determined for each cluster region.

5.2 TSV-Island Insertion

After running our clustering algorithm, we determined possible cluster regions (per net) where TSV islands can be inserted. However, not all clusters need to have TSV islands inserted to allow routing all nets through TSVs — according to the bound O_{link}, each net can be included in several clusters. Depending on the order of selecting clusters for TSV-island insertion, some clusters may become infeasible as TSV-island sites. Available deadspace accounted for a particular cluster may be occupied by another cluster. Furthermore, clusters containing nets which have not been clustered within unobstructed tiles need to consider nearby deadspace. This may also result in TSV islands blocking each other.

Our TSV-island insertion algorithm, illustrated in Figures 7 and 4, accounts for aligned deadspace while iteratively assigning nets to clusters and inserting TSV islands. In the following discussion, we refer to nets yet unassigned to a TSV island as *non-inserted*

```
INSERT_TSV_ISLANDS(C, N, T)
 1  // Phase 4: sort nets
 2  SORT_NETS_BY_AREA_SUPPLY(N, C)
 3  progress = TRUE
 4  while progress == TRUE
 5      // Phase 5: assign nets to clusters
 6      foreach net n ∈ N where n.inserted == FALSE
 7          c = FIND_HIGHEST_SCORED_CLUSTER(n, C, n^d_min)
 8          ASSIGN_NETS_TO_CLUSTER(c.nets, c, O_link)
 9      // Phase 6: iteratively insert TSV island for a largest cluster
10      (c, t) = INSERT_TSV_ISLAND(C, T)
11      progress = (c != NULL)
12      // Phase 6: mark & unlink handled nets from clusters
13      if progress == TRUE
14          foreach net m ∈ c.nets
15              m.inserted = TRUE
16              m.TSV_island = t
17              REMOVE_NET_FROM_CLUSTERS(m, C \ c)
18          // Phase 6: remove all assignments of nets to clusters
19          REMOVE_ASSIGNMENTS_FROM_CLUSTERS(C \ c)
20      elseif ∄ n ∈ N where n.inserted == FALSE
21          TERMINATE(success)
22      else TERMINATE(fail)
```

Figure 7: Our TSV-island insertion algorithm. Input data are described in Section 4.

nets, and to nets yet unassigned to a cluster as *unassigned nets*. **In Phase 4**, our algorithm sorts all nets by their total aligned deadspace of related clusters. Nets included in clusters with little available deadspace are considered first, since corresponding TSV islands are difficult to insert. **In Phase 5**, each unassigned net is analyzed for its associated clusters. The highest-scored cluster with respect to a dynamic *cluster score* Υ (available deadspace of cluster region divided by number of assigned nets) is chosen. Calculation and assignment of Υ for each cluster is performed dynamically within procedure FIND_HIGHEST_SCORED_CLUSTER, depending on previously assigned nets. In order to facilitate TSV-island insertion, the cluster to be chosen must provide a minimal amount of deadspace n^d_{min} for each net to be assigned to it. Then, each net of the highest-scored cluster is assigned to the cluster, subject to O_{link}. **In Phase 6**, TSV-island insertion for a largest cluster (in terms of assigned nets divided by Υ) is iteratively attempted — TSV-island insertion for clusters with many assigned nets and little available deadspace is difficult, thus these clusters are considered first. A local search (in procedure INSERT_TSV_ISLAND) over the cluster regions identifies contiguous regions with appropriate shapes to insert a TSV island with sufficient capacity.[4] Initially, aligned deadspace within the cluster regions only is considered. If no contiguous regions of deadspace can be found, a second iteration expands the cluster regions by factors c^x_{ext}, c^y_{ext} (in terms of die dimensions) to widen the search. If no contiguous regions are found again for any cluster, *block shifting* is performed to increase aligned deadspace (Section 5.3). Therefore, the cluster providing maximal amount of aligned deadspace is chosen first to minimize the total amount of shifting. After successful TSV-island insertion, all nets are unlinked from remaining clusters — according to Υ, each net may be assigned to different clusters now. Iterations continue with Phase 5 until all nets are inserted. If TSV-island insertion fails for all available clusters, our algorithm terminates with no solution.

5.3 Block Shifting using Floorplan Slacks

TSV-island insertion can fail because aligned deadspace is unavailable where it is needed. To address these failures, we propose to redistribute deadspace by shifting blocks in *x*- and/or *y*-directions

[4]The local search and TSV-island insertion are not described in detail due to page limitations.

without changing the floorplan outline or block ordering. We utilize the well-known concept of *floorplan slack* [1], which describes maximal displacement of a block within the floorplan. When blocks do not overlap, slacks are ≥ 0. We determine slacks for each layer separately and use standard linear-time traversals of floorplan constraint graphs, not unlike those in *Static Timing Analysis* [27]. Floorplan modifications based on constraint graphs are discussed in detail in [25]. To calculate *x*-slacks, we (i) pack blocks to the left boundary, and (ii) pack blocks to the right boundary. *x*-slack for each block is computed as the difference of the block's *x*-coordinates in these two packings. *y*-slack is calculated in the same way.

When TSV-island insertion requires redistributing deadspace by block shifting, we identify individual clusters in need of aligned deadspace. Within each cluster region, we determine the largest region R_d of aligned deadspace (if no aligned deadspace is found, we nominally consider the center of the cluster region as R_d). We then seek to consolidate additional aligned deadspace around R_d by shifting away the blocks adjacent to R_d. The distance by which each block is shifted cannot exceed its slack in the respective direction. Furthermore, the sum of such displacements in each direction cannot exceed the floorplan slack (the largest slack of any one block). Therefore, we shift blocks incrementally by small amounts so as to increase R_d until it reaches the size required to accommodate a TSV island with sufficient capacity.

Shifting a block may require shifting its abutting neighbors and other blocks. To this end, we maintain the floorplan configuration using constraint graphs and implement block shifting as follows. First, block dimension is inflated by the amount of displacement, and then standard path-tracing algorithms are applied to the constraint graph to find new block locations. To speed up this procedure, path-tracing can be performed incrementally, not unlike in incremental Static Timing Analysis.

If R_d cannot be increased sufficiently, we choose another region of aligned deadspace within the cluster region.

6. EMPIRICAL VALIDATION

We obtain 3D floorplans by running state-of-the-art software for 3D floorplanning.[5] The GSRC and MCNC benchmarks included in this infrastructure do not provide pin offsets, therefore we assume net bounding boxes to be defined by the bounding boxes of incident blocks. Two sets of floorplans are obtained, configuring the floorplanner for 10% and 15% deadspace respectively. We construct two sets of rectangular TSV islands with TSV footprints of $2\mu m^2$ and $4\mu m^2$ respectively. Each set contains TSVs islands with capacities for 2-30 nets; TSV islands are designed by packing single TSVs while considering keep-out-zone distances of $1\mu m$.

Experimental configurations. We consider two design configurations; one with guaranteed channels, one without channels. (Traditional floorplanners usually pack blocks without channels; however, many industry chips include channels between blocks to facilitate routing.) To insert channels between the blocks without modifying the floorplanner, every block was inflated before floorplanning and contracted to the original size after floorplanning. However, this increases floorplan size. An alternative is to pack blocks without channels, but carefully redistribute deadspace to facilitate TSV-island insertion. While more complex, this approach produces much more compact floorplans.

[5]We thank the authors of [32] and Yuankai Chen for sharing their infrastructure for 3D floorplanning.

Infrastructure for empirical validation. We implemented our algorithms using C++/STL, compiled them with g++ 4.4.4, and ran on a 64-bit Linux system with a 3.0GHz *Intel Core 2* processor and 8GB RAM. Parameters discussed in Section 5 are configured according to Table 1; Ξ_{min}^{d}, Ω_{min}, O_{nets}, and O_{link} control the clustering algorithm, while c_{ext}^{x} and c_{ext}^{y} control the deadspace search for TSV-island insertion, and i_{max}, f_{max}, i_{min}, and f_{min} control the global iterations. Experimental results on representative GSRC and MCNC benchmarks are reported in Table 2. In cases where our algorithm terminates with no solution, results are marked as *fail*.

TSV-island insertion and estimated wirelength. We analyze the impact of available TSV-island types, considering their capacity and dimensions. As expected, smaller TSVs increase chances of TSV-island insertion. However, wirelength overhead varies only marginally for the TSV-island dimensions that we tried. Being able to adjust the shape of TSV islands is beneficial for TSV-island insertion, while using square islands hinders insertion.

The overhead of TSV islands can be estimated by comparing actual net lengths to shortest paths (see *HPWL ratio* in Table 2), which would correspond to greedy insertion of single TSVs within net bounding boxes. Here we do not account for the increased footprint of single TSVs (due to increased keep-out-zones in comparison to packed TSV arrays) and have no way to quantify the loss of redundancy offered by TSV islands. Furthermore, this comparison would only apply to block-level interconnect, whereas most of the design's nets are subsumed within blocks. Wirelength overhead is 13.3-17.2% for the configuration without guaranteed channels, and 1.1-7.1% for the configuration with guaranteed channels. Such a moderate overhead in global interconnect, especially for the configuration with guaranteed channels, is expected and can be tolerated because 3D integration offers greater benefits [13].

Block shifting vs. guaranteed channels. Recall that inserting channels increases floorplan's deadspace. In contrast, redistributing available deadspace to facilitate TSV-island insertion supports more compact floorplans. However, we observe that floorplan slack required for block shifting is often insufficient, especially for blocks surrounding and/or covering cluster regions. Floorplans obtained with increased-deadspace configuration do not necessarily increase *aligned* deadspace because the infrastructure of [32] does not account for. For successful TSV-island insertion, our algorithms must

Deadspace & TSV area	Metrics	Design configuration					
		w/o guaranteed channels			w/ guaranteed channels		
		n100	*n200*	*n300*	*n100*	*n200*	*n300*
10% 2μm^2	HPWL ratio	1.134	1.167	1.172	1.011	1.044	1.059
	Channel size	-	-	-	8%	13%	8%
	Runtime (s)	22.73	147.38	392.54	4.61	40.67	102.04
15% 2μm^2	HPWL ratio	1.147	1.153	1.159	1.011	1.037	1.048
	Channel size	-	-	-	13%	12%	11%
	Runtime (s)	20.60	108.17	450.80	4.63	40.31	106.77
10% 4μm^2	HPWL ratio	1.133	fail	fail	1.027	1.071	1.070
	Channel size	-	-	-	10%	14%	12%
	Runtime (s)	13.45	fail	fail	5.10	53.51	111.62
15% 4μm^2	HPWL ratio	fail	fail	fail	1.011	1.066	1.062
	Channel size	-	-	-	10%	14%	12%
	Runtime (s)	fail	fail	fail	4.75	63.19	120.78

Table 2: L2Di integration without and with guaranteed channels. To add channels, every block was inflated before floorplanning and contracted to the original size after floorplanning, increasing floorplan outline. Binary search was used to find solutions with minimal wirelength overhead. We report channel size as the inflation factor. HPWL ratio divides wirelength of connections through TSVs by shortest-path wirelengths.

use aligned deadspace further from their cluster regions, thus increasing wirelength overhead.

Inserting deadspace channels not only reduces wirelength overhead and runtime, but also helps L2Di integration succeed where it otherwise fails. We performed a binary search for channel insertion with block inflations ranging from 6% to 14%. Considering the trade-off between floorplan increase and wirelength overhead, our results represent lowest wirelength overheads. Figures 8(a,b) illustrate L2Di integration of *n300* without and with guaranteed channels respectively. Some TSV islands are placed outside cluster regions due to deadspace limitations within cluster regions, introducing wirelength overhead. Cluster regions differ for the two design configurations. Without guaranteed channels, less aligned deadspace is available in general. Since our clustering algorithm accounts for available deadspace, cluster regions are more likely to intersect in this configuration.

7. CONCLUSION

Our work seeks to streamline the transition from existing practice in chip design to 3D integration. In addition to manufacturing and cost considerations, this transition is hampered by the lack of relevant standards and commercial EDA tools. A key insight in our work is that many of the benefits of 3D integration can be obtained while reusing existing 2D blocks. Therefore, we analyzed different design styles for 3D integration of 2D blocks. We conclude that, in the near future, the most promising and least risky design style for 3D integration is the L2Di style.

To enable the L2Di style, we contribute novel techniques for clustering of nets and inserting TSV islands. We provide an empirical validation of our techniques, demonstrating the possibility for 3D integration of 2D blocks without modifying their layout. We observe that the use of TSV islands may increase wirelengths beyond shortest paths, depending on the given floorplan. However, this effect can be mitigated by using a larger number of (smaller) TSV islands and by inserting deadspace channels between blocks.

Metric	Meaning	Value
Ξ_{min}^{d}	Min aligned deadspace per clustering-grid tile (*tile size*)	0.9
Ω_{min}	Min overlap area between cluster region and grid tile (*tile size*)	0.25
O_{nets}	Max nets per cluster	30
O_{link}	Max clusters per net	5
n_{min}^{d}	Min aligned deadspace per net in a cluster (*TSV footprint*)	1.05
c_{ext}^{x}, c_{ext}^{y}	Factors for extending cluster region to search for nearby deadspace (*die dimensions*)	0.05
bb_{min}	Area of the smallest net bounding box	floorplan specific
i_{max}	Global iterations for decreasing tile size from f_{max} to bb_{min}	5
f_{max}	Max tile size (bb_{min})	1.05
i_{min}	Global iterations for decreasing tile size from bb_{min} to f_{min}	35
f_{min}	Min tile size (bb_{min})	0.6

Table 1: Parameters for net clustering and TSV-island insertion algorithms, along with their values.

References

[1] S. N. Adya, I. L. Markov. Consistent placement of macro-blocks using floorplanning and standard-cell placement. *ISPD '02*, pp. 12–17.

[2] J. Cong, Y. Ma. Thermal-aware 3D floorplan. *Three Dimensional Integrated Circuit Design*, pp. 63–102. Springer US, 2010.

[3] J. Cong, J. Wei, Y. Zhang. A thermal-driven floorplanning algorithm for 3D ICs. *ICCAD '04*, pp. 306–313.

[4] C. Ferri, S. Reda, R. I. Bahar. Strategies for improving the parametric yield and profits of 3D ICs. *ICCAD '07*, pp. 220–226.

[5] R. Fischbach, J. Lienig, T. Meister. From 3D circuit technologies and data structures to interconnect prediction. *SLIP '09*, pp. 77–84.

[6] R. J. Fowler, M. S. Paterson, S. L. Tanimoto. Optimal packing and covering in the plane are NP-complete. *IPL*, 12(3):133–137, 1981.

[7] S. Garg, D. Marculescu. 3D-GCP: An analytical model for the impact of process variations on the critical path delay distribution of 3D ICs. *ISQED '09*, pp. 147–155.

[8] M. C. Golumbic. *Algorithmic graph theory and perfect graphs*. Elsevier, 2004.

[9] M. B. Healy et al. Design and analysis of 3D-MAPS: A many-core 3D processor with stacked memory. *CICC '10*.

[10] A.-C. Hsieh et al. TSV redundancy: Architecture and design issues in 3D IC. *DATE '10*, pp. 166–171.

[11] W.-L. Hung et al. Interconnect and thermal-aware floorplanning for 3D microprocessors. *ISQED '06*, pp. 98–104.

[12] H. Imai, T. Asano. Finding the connected components and a maximum clique of an intersection graph of rectangles in the plane. *J. Algorithms*, 4(4):310–323, 1983.

[13] International technology roadmap for semiconductors. http://www.itrs.net/Links/2009ITRS/Home2009.htm.

[14] L. Jiang, Q. Xu, K. Chakrabarty, T. M. Mak. Layout-driven test-architecture design and optimization for 3D SoCs under pre-bond test-pin-count constraint. *ICCAD '09*, pp. 191–196.

[15] D. H. Kim, K. Athikulwongse, S. K. Lim. A study of through-silicon-via impact on the 3D stacked IC layout. *ICCAD '09*, pp. 674–680.

[16] D. H. Kim, S. Mukhopadhyay, S. K. Lim. Through-silicon-via aware interconnect prediction and optimization for 3D stacked ICs. *SLIP '09*, pp. 85–92.

[17] H.-H. S. Lee, K. Chakrabarty. Test challenges for 3D integrated circuits. *IEEE Design & Test of Computers*, 26(5):26–35, 2009.

[18] Y.-J. Lee, R. Goel, S. K. Lim. Multi-functional interconnect co-optimization for fast and reliable 3D stacked ICs. *ICCAD '09*, pp. 645–651.

[19] Y.-J. Lee, M. Healy, S. K. Lim. Co-design of reliable signal and power interconnects in 3D stacked ICs. *IITC '09*, pp. 56–58.

[20] X. Li et al. LP based white space redistribution for thermal via planning and performance optimization in 3D ICs. *ASP-DAC '08*, pp. 209–212.

[21] X. Li, Y. Ma, X. Hong. A novel thermal optimization flow using incremental floorplanning for 3D ICs. *ASP-DAC '09*, pp. 347–352.

[22] Z. Li et al. Integrating dynamic thermal via planning with 3D floorplanning algorithm. *ISPD '06*, pp. 178–185.

[23] Z. Li et al. Efficient thermal-oriented 3D floorplanning and thermal via planning for two-stacked-die integration. *TODAES*, 11(2):325–345, 2006.

[24] I. Loi et al. A low-overhead fault tolerance scheme for TSV-based 3D network on chip links. *ICCAD '08*, pp. 598–602.

[25] M. D. Moffitt, J. A. Roy, I. L. Markov, M. E. Pollack. Constraint-driven floorplan repair. *TODAES*, 13(4):1–13, 2008.

[26] C. H. Papadimitriou, K. Steiglitz. *Combinatorial Optimization : Algorithms and Complexity*. Dover Publications, 1998.

[27] S. S. Sapatnekar. *Timing*. Kluwer, 2004.

[28] S. Sridharan et al. A criticality-driven microarchitectural three dimensional (3D) floorplanner. *ASP-DAC '09*, pp. 763–768.

[29] E. Wong, S. K. Lim. Whitespace redistribution for thermal via insertion in 3D stacked ICs. *ICCD '07*, pp. 267 –272.

[30] X. Wu et al. Cost-driven 3D integration with interconnect layers. *DAC '10*, pp. 150–155.

[31] J.-S. Yang et al. TSV stress aware timing analysis with applications to 3D-IC layout optimization. *DAC '10*, pp. 803–806.

[32] P. Zhou et al. 3D-STAF: scalable temperature and leakage aware floorplanning for three-dimensional integrated circuits. *ICCAD '07*, pp. 590–597.
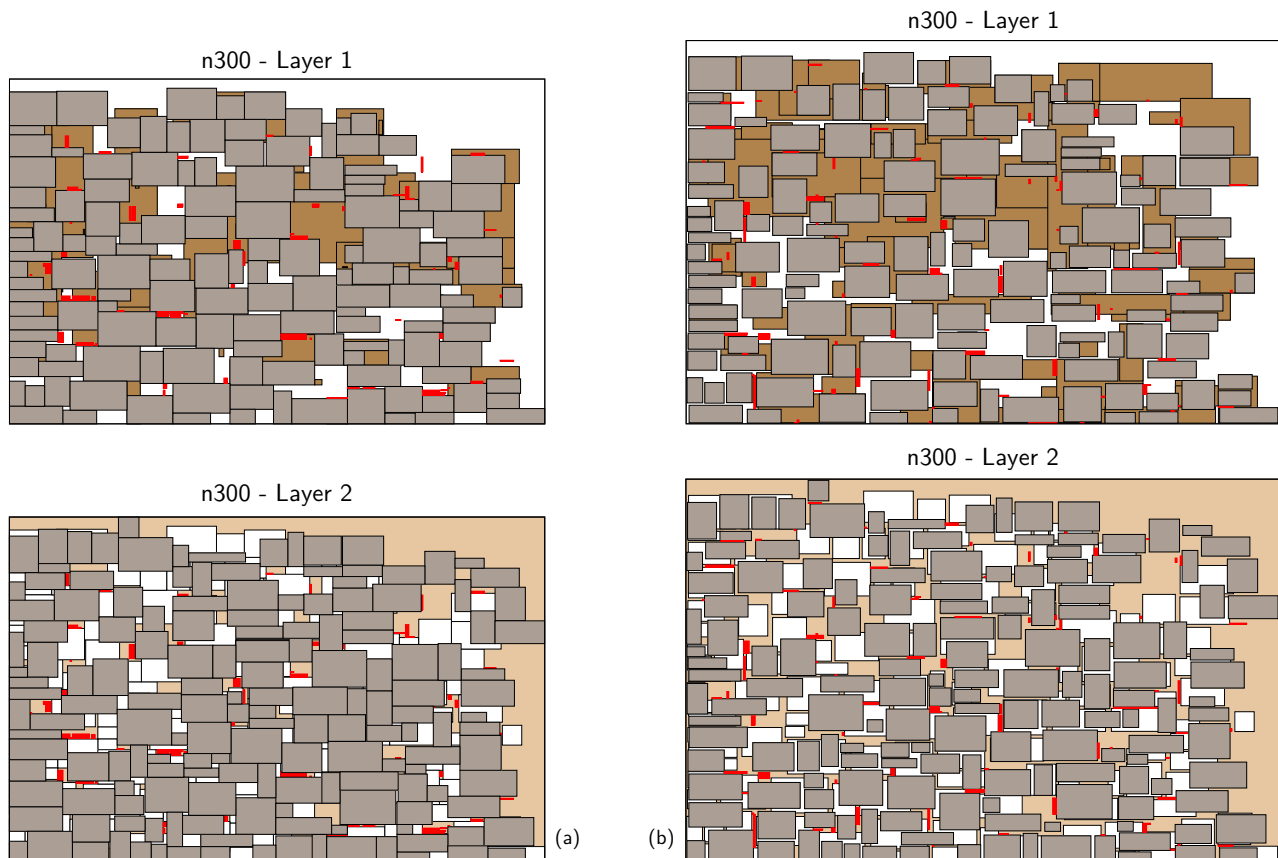
Figure 8: L2Di-style integration of *n300*, (a) without guaranteed channels and (b) with guaranteed channels. Grey rectangles represent blocks, red (black) rectangles TSV islands (up to 30 TSVs per island). Cluster regions are shown in brown (dark grey) on Layer 1, while aligned deadspace is shown in beige (light grey) on Layer 2. Channel insertion in (b) increased die size by 21%.