Structural Planning of 3D-IC Interconnects by Block Alignment

Johann Knechtel*, Evangeline F. Y. Young**, and Jens Lienig*
* Institute of Electromechanical and Electronic Design, Dresden University of Technology, Germany
** Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong
johann.knechtel@ifte.de, fyyoung@cse.cuhk.edu.hk, jens@ieee.org

Abstract—Three-dimensional integrated circuits rely on optimized interconnect structures for blocks which are spread among one or multiple dies. We demonstrate how 2D and 3D block alignment can be efficiently utilized for structural planning of different interconnects. To realize this, we extend the corner block list and provide effective techniques for 3D layout generation, i.e., block placement and alignment. Our techniques are made available in an open-source, simulated-annealing-based tool. Besides block alignment, it accounts for key objectives in 3D design like fast thermal management and fixed-outline floorplanning. Experimental results on GSRC and IBM-HB+ circuits demonstrate the capabilities of our tool for both planning 3D-IC interconnects by block alignment and for 3D floorplanning in general.

I. Introduction

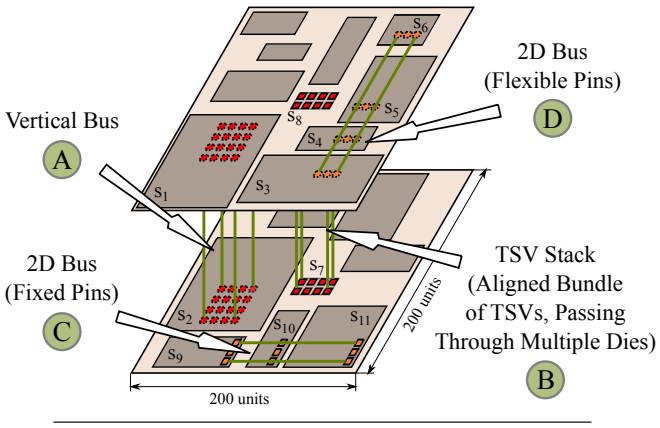
Three-dimensional (3D) stacking of active dies is recognized as a promising approach to meet demands on todays' and future chips regarding their performance, functionality and power consumption. Vertical plugs connecting through separate dies, mainly the through-silicon vias (TSVs), facilitate short and low-power interconnects and thus enable high-performance 3D integrated circuits (3D ICs). 3D network on chip (NoC) architectures have been proposed to increase communication capabilities for logic integration [1] or memory integration [2]. Complementing such approaches, the well-known concept of *bus planning*, i.e., grouping multiple signals into adjacent wires, remains also relevant for 3D-IC design.

Although the concept of *block alignment* has been successfully applied in 2D layout representations for bus planning [3, 4], it has been every so often neglected in 3D representations. Some studies, e.g., [5–7], enable *fixed alignment*, i.e., blocks are to be aligned (possibly across several dies) such that their relative positions fulfill fixed distances. However, an application to vertical-bus planning is only indicated in [6]. To the best of our knowledge, none of the existing studies considers *alignment ranges*, i.e., blocks are to be aligned such that their relative positions fulfill upper and/or lower distance limits. Thus, "flexible" block alignment is not supported so far. We observe that utilizing these different alignment approaches enables structural planning of interconnects for 3D ICs—as illustrated in Figs. 1 and 2, processing block alignment allows one to design dedicated, straight interconnect structures.

To address previously inadequate support for such interconnect structures during 3D floorplanning, we present a methodology based on orchestrated block placement and alignment. In our study, we consider interconnects for different practical scenarios in 3D ICs, as further motivated in Section II.

Our contributions can be summarized as follows.

1. We propose an extension of the corner block list (CBL) (Section IV). Our extension can (i) encode both fixed alignment and alignment ranges, as well as (ii) handle inter- and intra-die alignment in a unified manner. (See Section II for these terms and related background.)
2. We develop effective techniques for 3D layout generation, i.e., block placement and alignment as well as layout packing (Subsection IV-B).
3. We provide an open-source 3D-floorplanning tool based on our CBL extension and simulated annealing (SA) (Section V). Besides block alignment, our tool considers these key objectives: fixed outlines, fast thermal management, layout packing, and wirelength optimization.



Alignment Encoding (Subsection IV-A):

(A) $a_1 = (s_1, s_2, (40, 1), (40, 1))$

(B) $a_2 = (s_7, s_8, (0, 0), (0,0))$

(C) $a_3 = (s_9, s_{10}, (150, 2), (0, 0))$,
$a_4 = (s_9, s_{11}, (150, 2), (0, 0))$

(D) $a_5 = (s_3, s_4, (30, 1), (80, 2))$,
$a_6 = (s_4, s_5, (30, 1), (80, 2))$,
$a_7 = (s_5, s_6, (30, 1), (80, 2))$

Fig. 1. Interconnect structures in a 3D IC and related block-alignment configurations. Vertical buses (A) are essential to connect (split-up) blocks among adjacent dies. TSV stacks (B) comprise aligned bundles of TSVs, are passing two or more dies, and are for example used in 3D NoCs. Both interconnect structures rely on *inter-die alignment*, i.e., blocks spread among several dies are to be aligned. Regular 2D buses with fixed or flexible pins (C, D) are traditionally considered to optimize datapaths or similar structures; they require blocks to be aligned within one die, i.e., rely on *intra-die alignment*.

II. Bus and Via Structures in 3D ICs and Related Block Alignment

As for the 3D design style, we consider block-level integration of 2D blocks. This style is acknowledged as a reliable and efficient approach, especially for first commercial 3D-IC applications [8, 9]. In such 3D ICs, routing paths for (massively parallel) interconnect structures can be enabled by means of block alignment (Figs. 1, 2). Block alignment in 3D ICs can be generally classified into *inter-die alignment*, i.e., blocks spread

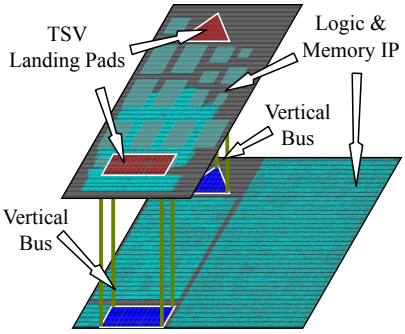


Fig. 2. Die shots of two macroblocks, partitioned across adjacent dies for delay and power optimization [14]. Embedded vertical buses require the macros to be aligned such that TSVs and related landing pads can be included.

among several dies are to be aligned, and *intra-die alignment*, i.e., blocks are aligned within one die. The variety of alignment specifics arise from different scenarios for 3D integration and interconnects, which are reviewed next. Note that we focus on signal interconnects in this work. For optimized planning of other interconnect types refer to, e.g., [10].

Monolithic integration has recently gained more interest due to advances in manufacturing processes; for block-level integration, this technology is advantageous in terms of improved interconnectivity [11]. In the general context of massively-interconnected dies, planning *vertical buses* which connect particular (split-up) blocks spread on separate dies is critical and should thus be considered from early design phases on. It is important to note that TSV-based integration can also exploit such buses, assuming that blocks can be adapted to include TSVs. For example, consider the two macroblocks in Fig. 2: this arrangement of tightly interconnected (for delay and power consumption optimized) modules relies on vertical buses, which are implemented by groups of TSVs. Accounting for such vertical buses during floorplanning requires capabilities for inter-die alignment. That is, in order to include a large number of vertical interconnects, the related blocks have to exhibit some intersecting regions.

A special case of vertical buses are *aligned TSV stacks*, i.e., TSVs are grouped and placed such that straight interconnects are passing through multiple dies. TSV stacks are relevant for different applications, e.g., to realize regular 3D NoCs, or to limit power-supply noise and to improve thermal distribution [12,13]. Consideration of aligned TSV stacks during floorplanning requires inter-die alignment with fixed offsets.

Regular (2D) bus structures connecting blocks within dies are independent of the 3D-integration technology. These buses are traditionally considered for several scenarios, e.g., to optimize datapath interconnects. Note that such structures rely on intra-die alignment of related blocks. Depending on fixed / flexible block pins, the planning of 2D buses requires support for fixed alignment / alignment ranges.

III. Basic Principles of Corner Block List

The corner block list (CBL) [15] is a topological 2D layout representation. In our work, we utilize it mainly for its efficiency (layout generation has a $\mathcal{O}(n)$ complexity) and feasible expandability towards a 3D representation (Section IV).

The CBL encodes a floorplan solution as tuple (S, L, T) where S is the *block-insertion sequence*, L the *insertion-direction sequence*, and T the *sequence of covered T-junctions* (see next paragraph). Note that conceptual *rooms*, i.e., dimensionless entities, are encoded in S. Each block is associated with a room; to obtain the physical layout, a transformation from the room topology to block coordinates is required.

During sequential layout generation, two criteria are to be considered for each block (within a room) $s_i \in S$: first, the *insertion direction* where $l_i = 0$ encodes vertical placement and $l_i = 1$ horizontal placement, respectively; second, the number t_i of *T-junctions* to be covered. The notion of T-junctions is a verbatim encoding; for example, $t_i = 1$ requires to (perpendicularly) cover the common boundary of two adjacent blocks.

IV. Corblivar: Corner Block List for Varied Alignment Requests

To enable interconnect structures during 3D floorplanning, we propose an extension of the classical 2D CBL. Our extension is named *corner block list for varied alignment requests (Corblivar)*. It encodes a 3D-IC design integrated on n dies using an ordered sequence $\{CBL_1, \ldots, CBL_n\}$ of CBL tuples and one global *alignment sequence* A. (Thus, Corblivar is a so-called 2.5D layout representation. Refer to [16] for an investigation of several previous 2.5D and 3D representations.) The *alignment tuples* $a_k \in A = \{a_1, \ldots, a_n\}$ are designed to encode different types of alignment requests as defined below and illustrated in Fig. 1. Like any layout representation, we need to embed Corblivar in a floorplanning tool; core parts and main features are outlined in Fig. 3.

A. Alignment Tuples

Definition of alignment tuples – Assume the placement of block s_j has to consider some alignment request with regard to (w.r.t.) s_i. The request is then defined as tuple $a_k = (s_i, s_j, (AR_x, ART_x), (AR_y, ART_y))$ where (AR_x, ART_x) and (AR_y, ART_y) denote the partial requests with respect to the x- and y-coordinate. These requests can be *independently* defined as *fixed offset* ($ART = 0$), as *minimal overlap* ($ART = 1$), as *maximal distance* ($ART = 2$) or as *don't care* ($ART = -1$); the meaning of these types is explained next.

Alignment types – Given a *fixed offset*, s_j is to be placed AR_x/AR_y units to the right/top ($AR_x/AR_y \geq 0$) or to the left/bottom ($AR_x/AR_y < 0$) of s_i, respectively, w.r.t. the blocks' lower-left corners. Fixed-offset alignment is required for restricted placement, e.g., of blocks with fixed pins.

For a (positive) *minimal overlap*, the projected intersection of blocks s_i and s_j must be *at least* AR_x units wide and/or AR_y units high. The intention of such alignment is to ensure straight but locally flexible paths for subsequent bus routing / placement of vertical interconnects.

An alignment request defining a *maximal distance* requires that the center points of blocks s_i and s_j are *at most* AR_x/AR_y units apart. This way, interconnects structures can be easily limited in their length and/or width.

It may not be necessary to define a request for both x- and y-coordinates; we label the unrestricted coordinate's request simply as *don't care*.

Dynamic interpretation of requests – Note that the introduced tuples can be easily utilized for both intra- or inter-die alignment by assigning related blocks to one common or to separate CBLs (dies). In other words, the proposed encoding does not restrict blocks to particular dies. For requests spanning multiple blocks (discussed next), it is also possible to combine intra- and inter-die alignment for several blocks.

Definition of tuples to align multiple blocks – For 3D-IC interconnects, implementing links among multiple blocks

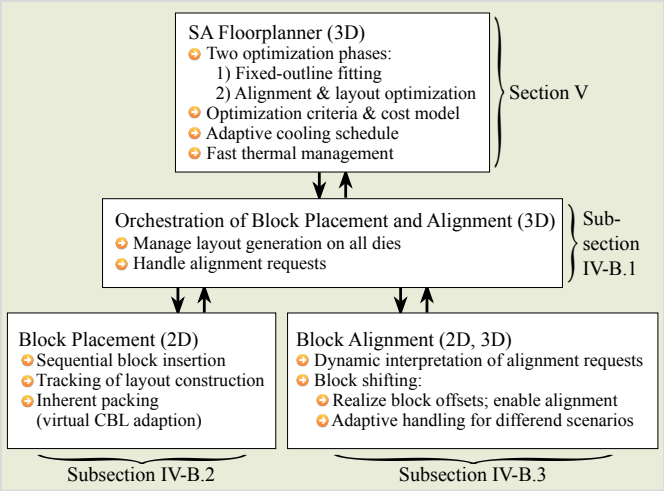


Fig. 3. Corblivar's components, embedded in a SA-based floorplanning tool. *Orchestration of Block Placement and Alignment* interacts with the SA heuristic for layout optimization, monitors the overall layout process, and delegates to *Block Placement* and *Block Alignment* in a synchronized manner.

is an essential scenario. Thus, let us assume the placement of blocks $s_1 \dots s_n$ has to consider several, combined alignment requests for interconnects planning. The required set of tuples can be derived in any desired fashion. For example, for requests requiring *one reference block* s_1 (e.g., to represent one specific end of a bus), the tuples would be defined as $(s_1, s_2, (AR_{x_2}, ART_{x_2}), (AR_{y_2}, ART_{y_2}))$, $\dots\dots\dots\dots\dots$, $(s_1, s_n, (AR_{x_n}, ART_{x_n}), (AR_{y_n}, ART_{y_n}))$. To give another example, we can encode alignments in a chain-like fashion to enable flexible interconnect structures (i.e., allowing local deviations from a straight, global path): $(s_1, s_2, (AR_{x_2}, ART_{x_2}), (AR_{y_2}, ART_{y_2}))$, $(s_2, s_3, (AR_{x_3}, ART_{x_3}), (AR_{y_3}, ART_{y_3}))$, $\dots\dots\dots\dots\dots$, $(s_{n-1}, s_n, (AR_{x_n}, ART_{x_n}), (AR_{y_n}, ART_{y_n}))$.

B. Layout Generation

We extend the CBL technique [15] in order to (i) handle inter- and intra-die alignment simultaneously, (ii) consider fixed offsets as well as alignment ranges, and (iii) perform effective layout packing. In the following subsections, we first discuss the orchestration of block placement and alignment and then provide techniques for these steps themselves.

B.1 Orchestration of Block Placement and Alignment

We next discuss the overall process of 3D layout generation. As illustrated in Fig. 3, this requires to (i) manage the layout-generation progress on all dies, (ii) handle the alignment requests, and (iii) interact with block placement and alignment (Subsections B.2 and B.3). In the following, we label "calls" to latter techniques as PLACE and ALIGN, respectively.

Auxiliary data structures – We memorize alignment requests in progress using the *alignment stack* AS. *Progress pointers* $p_i = s_j$ denote the currently processed block s_j for each die d_i. A *die pointer* $p = d_i$ is used to keep track of the currently processed die.

Process flow (Algorithm 1) – We perform the following steps for each block s_i. Initially, we check whether the associated die d is currently marked as *stalled* (line 5), i.e., layout generation is halted due to another alignment request in progress—this occurs for *intersecting requests*, i.e., related blocks are arranged in the CBL sequences such that their placement is interfering. To resolve this, we need to unlock die d—we PLACE the current block s_i, mark related changes, and proceed with the next block (lines 6–9). Otherwise (for non-stalled dies), we check if some alignment requests a_k are applying to s_i (line 11). If no a_k are found, we directly PLACE s_i and proceed with the next block (lines 28–29). If some request(s) a_k are defined, we need to handle them appropriately (lines 12–23), as described next. For any given a_k, we search the stack AS for it and continue accordingly. *Case a:* if a_k is found, it was previously handled while processing s_j, that is the block to be aligned with s_i. Thus, it is assured that preceding blocks on both related dies are placed at this point. We can now safely ALIGN both s_i and s_j, mark them as placed, and drop the request a_k (lines 14–17). Note that only in cases where *all* requests for s_i are handled, we proceed on the current die d (line 25). Otherwise, we continue layout generation without loss of generality (w.l.o.g.) on s_j's die d' (line 21). *Case b:* if a_k is not found in AS, s_j was not processed yet. We then memorize a_k as in progress, halt layout generation on d, and continue on d' (lines 19–21). Finally, if layout generation is done on d, we proceed on yet unfinished dies until the whole 3D layout is generated (lines 32–38).

Be aware that *deadlock situations*, i.e., layout generation on different dies is waiting for each other until particular blocks can be aligned, *cannot* occur due to resolving of stalled dies. This is true for any alignment request; see also Subsection B.3 for implications on block alignment.

B.2 Block Placement

To maintain a valid layout during placement, it is necessary to consider previously placed blocks. We propose a technique which allows us to (i) efficiently keep track of relevant blocks, (ii) fix CBL tuples w.r.t. exceeding T-junctions, and (iii) virtu-

Algorithm 1 Orchestration of Block Placement and Alignment

```
1:  p ← d_1                                   ▷ start without loss of generality on bottom die
2:  p_i ← s_1
3:  loop
4:      s_i ← p_i ← p
5:      if die d ← p is stalled then
6:          PLACE(s_i)
7:          mark s_i in any a_i' ∈ A as placed
8:          mark d as not stalled
9:          p_i ← p_{i+1}
10:     else                                                          ▷ d is not stalled
11:         if some a_k are defined for s_i then
12:             for all a_k do                    ▷ consider a_k w/ placed blocks first
13:                 if a_k in AS then
14:                     ALIGN(a_k)
15:                     remove a_k from AS
16:                     mark s_i, s_j in any a_i' ∈ A as placed
17:                     mark d_j = die(s_j) as not stalled
18:                 else
19:                     add a_k to AS
20:                     mark d as stalled
21:                     p ← die(s_j)
22:                 end if
23:             end for                                               ▷ all a_k considered
24:             if d is not stalled then
25:                 p_i ← p_{i+1}
26:             end if
27:         else                                                      ▷ no a_k defined for s_i
28:             PLACE(s_i)
29:             p_i ← p_{i+1}
30:         end if
31:     end if                                                        ▷ s_i processed
32:     if p_i = end then
33:         if some p_j ≠ end then
34:             p ← p_j
35:         else
36:             return done
37:         end if
38:     end if
39: end loop
```

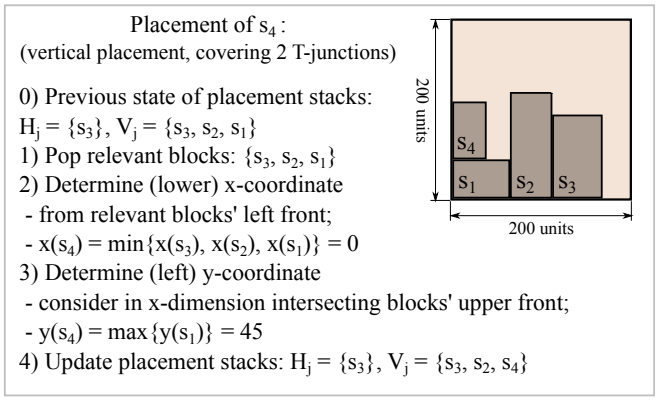


Fig. 4. Placement steps, to be performed for exemplary vertical insertion of block s_4 while covering 2 T-junctions.

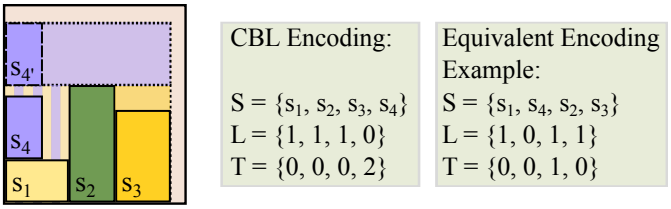


Fig. 5. Implications of virtual CBL adaption. Rooms and their assigned blocks are similarly colored. Block s_4 is placed "into the room" of s_1, thereby enabling a more compact layout. Without applying virtual CBL adaption, s_4 would be placed as $s_{4'}$. Virtual CBL adaption can result in different, equivalent encodings for the same, compact layout; hence, it supports an efficient solution-space exploration towards compact layouts.

ally adapt CBL tuples for implicit layout compaction. Our approach differs from [15] in these features but follows the same principle of sequential block placement into dissected rooms.

Auxiliary data structures – We keep track of placed blocks using two stacks H_j / V_j for each CBL_j. More precisely, these stacks are governed to contain CBL_j's blocks currently covering the vertical right / horizontal upper front of die d_j; these are considered as the *boundary fronts* for further placement.

Placement flow – We determine each blocks' $s_i \in S_j$ lower-left coordinates (x_i, y_i) as follows. (See Fig. 4 for an example.) *First*, we retrieve $t_i + 1$ previously placed blocks from the respective stacks H_j / V_j. These blocks are referred to as *relevant blocks* in the following. Note that in cases where only $t_{max} < t_i + 1$ blocks are available, the related CBL tuple is technically infeasible [15]. To fix such invalid tuples, we simply consider all t_{max} blocks in order to fulfill the desired covering of T-junctions as best as possible. *Second*, we determine s_i's y / x-coordinates (orthogonal to the horizontal / vertical insertion direction) by considering the structural change of CBL_j's room dissection. In case a new column / row is implicitly defined due to covering all relevant blocks during placement of s_i, we set the respective y / x-coordinate to 0. In the remaining cases, we derive the coordinate from the relevant blocks' lower / left front. This can be also thought of as placing a new column / row into the existing room dissection. *Third*, we determine s_i's x / y-coordinates (along the insertion direction) by considering the right / upper front of *previously placed blocks* which are intersecting with s_i in its orthogonal, recently determined y / x-coordinates. *Fourth*, we update the placement stacks to follow the changed layout's fronts as follows. We push s_i onto the insertion-direction-related stack H_j / V_j. In case s_i is not covered by some relevant block to its top / right front, we also push s_i to the (unrelated) stack V_j / H_j. Finally, we push relevant blocks not covered by s_i back to H_j / V_j; these blocks remain part of the layout's boundary front and are thus to be furthermore considered.

Virtual CBL adaption – For any block smaller than the room it is supposed to cover, the next, adjacent block(s) will be packed "into the room" of this smaller block (Fig. 5). We refer to this feature as virtual CBL adaption since it results in practice in different CBL tuples encoding the same (compact) layout. Note that virtual CBL adaption is generally applied during block placement.

B.3 Block Alignment: Inter- and Intra-Die Alignment

Recall that our alignment tuples support different alignment types and can be interpreted as inter- or intra-die requests. We observe that such diverse requests all depend on their assigned blocks' planar offsets, i.e., relative distances considering their projections onto a plane. This implies that we can rely on adjusting the blocks' offsets in order to handle alignment requests. Such adjustments are practical since our layout-generation process is synchronized across the whole 3D IC, i.e., blocks to be aligned "wait for each other's die to be ready", that means until preceding blocks are placed. This "waiting" might result in circular dependencies; layout generation handles such cases by resolving stalled dies (Subsection B.1).

It is also important to note that, depending on particular alignment and CBL configurations, it may be infeasible to fulfill all requests.[1] One resolution (exclusively applying) for failing *intra-die* alignments includes preprocessing CBL tuples and adjusting topologically infeasible configurations [17]. Yet, such preprocessing is not warranted in the presence of different alignment requests—the applicability of *inter-die* alignments depends on the layout of all dies, that is on the entire layout-generation process. The flow described below includes layout-aware techniques, i.e., enables alignment in some cases of previously placed blocks.

Alignment flow – Remember that alignment tuples always cover two blocks; requests spanning more blocks (and tuples) are thus (implicitly) handled stepwise. Initially, we need to check whether one or both blocks have been previously placed. Depending on preceding placement, three different scenarios are to be distinguished for handling request a_k.

Scenario I: both blocks are placed – In this case, we cannot fulfill a_k since we omit post-placement shifting.[2]

Scenario II: one block is yet unplaced – Here, we assume w.l.og. that $s_i \in a_k$ is yet unplaced and $s_j \in a_k$ was previously placed. Depending on both the coordinates of (placed) s_j and the properties of a_k, we may be able to fulfill a_k as follows. First, we determine s_i's y / x-coordinates orthogonal to its horizontal / vertical insertion direction ("Second", Subsection B.2). Next, based on both the inherent offset between s_i and s_j and the defined alignment of a_k, we derive the *required shifting range* $rs_y(s_i, s_j)$ / $rs_x(s_i, s_j)$, i.e., the remaining offset of s_i w.r.t. s_j in order to fulfill a_k. Note that $rs(s_i, s_j) = -rs(s_j, s_i)$, i.e., the shifting range is directed and invertible. In cases $rs(s_i, s_j) = 0$, a_k is already fulfilled. In cases $rs(s_i, s_j) < 0$, we would need to shift s_i downward / leftward which is trivially prohibited while maintaining a valid layout. Alternatively, we could shift placed s_j upward / rightward;

[1] We would like to stress that this limitation only applies to particular configurations. That means, adapting the CBL configurations during alignment-aware 3D floorplanning (as proposed in Section V) can resolve this issue.

[2] Based on our observations, shifting placed blocks most likely requires adjacent (or nearby) blocks to be shifted as well in order to maintain a valid layout. This is impractical in the presence of different alignment requests—such shifting can then undermine handling of remaining requests and even invalidate previously processed ones.

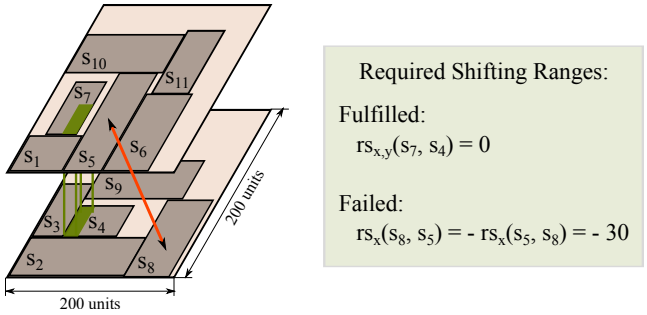


Fig. 6. Examples for required shifting ranges. During alignment of block s_7 with previously placed s_4, shifting s_7 to the right was applicable such that $rs(s_7, s_4)$ is resolved. In contrast, we cannot shift previously placed s_5 (in order to align it with s_8) since we omit post-placement shifting. Also, the inverted shifting range $rs(s_8, s_5) = -rs(s_5, s_8)$ cannot be resolved since this would require a shift of s_8 to the left, which is hindered by s_2.

however, this is not applicable, as mentioned before. (See also Fig. 6 for illustration.) Third, if and only if $rs_{y/x}(s_i, s_j) \geq 0$, we can perform a corresponding forwards shift of s_i in y / x-direction and thus satisfy a_k's partial request. Next, we perform above steps similarly for s_i's x / y-coordinates along its insertion direction. Finally, we handle the placement stacks. In case block shifting was conducted, we need to *rebuild* them, i.e., the stacks' current blocks along with s_i are sorted by their coordinates in descending order; afterwards uncovered (*relevant*) blocks redefine the stack. In case block shifting was not required, we simply update the stacks (Subsection B.2).

Scenario III: both blocks are yet unplaced – We are free to shift both blocks, thus we can satisfy a_k. Depending on the blocks' insertion direction and coordinates, we process block shifting in parallel or sequentially (similar as described above).

V. 3D-Floorplanning Tool

We provide Corblivar along with our C++ implementation of a SA-based 3D-floorplanning tool [18]. Our holistic concept of (orchestrated) block placement and alignment differs notably from previous works. Applying SA-based floorplanning during initial experiments, we observe limitations of existing techniques w.r.t. solution-space exploration for block alignment as well as for "classical" 3D floorplanning. Thus, some effective extensions are needed; notable features of our tool are (i) a SA framework including two optimization phases and specific cost models, (ii) an adaptive SA cooling schedule, and (iii) a fast yet sufficiently accurate thermal analysis.

A. Optimization Criteria & Cost Models

We next discuss applied optimization criteria along with their cost functions/models. Note that cost functions are formulated for SA's classical cost-minimization approach, i.e., lower cost correspond to more optimized layouts.

Outline – This criteria unifies evaluation of the layout's bounding boxes, i.e., packing density, as well as fixed-outline fitting. This is achieved by extending Chen and Chang's aspect-ratio-based cost model [19]. Our model is defined as

$$\begin{array}{lcl} c_{OL} & = & c_{PD} + c_{ARV} \\ c_{PD} & = & \frac{1}{2}\alpha(1 + \frac{n_{feasible}}{n}) \times \max_{d_i}\left(\frac{A_{outline}(b \in d_i)}{A_{outline}(d_i)}\right) \\ c_{ARV} & = & \frac{1}{2}\alpha(1 - \frac{n_{feasible}}{n}) \times \max_{d_i}\left(\Delta_{AR}(d_i)^2\right) \\ \Delta_{AR}(d_i) & = & AR_{outline}(b \in d_i) - AR_{outline}(d_i) \end{array}$$

where c_{PD} and c_{ARV} denote the respective cost terms for *packing density* and *aspect-ratio violation*. Functions $A_{outline}$ and $AR_{outline}$ determine the outline's area and aspect ratio w.r.t. a set of blocks $b \in d_i$ / a die d_i, respectively. Note that we perform cost calculation for previous n layout operations where $n_{feasible} \leq n$ operations resulted in a valid layout, i.e., blocks on all dies are fitting into the fixed outline.

Wirelength and TSV count – We assume that the lowermost die d_0 is connected to the package board. For each net n, we determine the half-perimeter wirelength (HPWL) on each related die d_i separately, denoted as $HPWL(n, d_i)$. To do so, we construct the bounding box by encircling connected terminal pins (only for d_0) and assigned blocks on d_i and on the upper die $d_j, j > i$ as well. The latter is required to model wires for connecting blocks with TSV landing pads in upper dies. Overall cost terms are defined as

$$\begin{array}{lcl} c_{WL} & = & \sum_n \left(l_{TSV} \times TSVs(n) + \sum_{d_i \in n} HPWL(n, d_i)\right) \\ c_{TSVs} & = & \sum_n TSVs(n) \end{array}$$

where $TSVs(n)$ denotes the required TSV count for net n. Note that we also account for "TSV wirelength" l_{TSV} in c_{WL}.

Thermal management – Cost c_T is the *estimated maximal temperature* of the critical die furthest away from the heatsink. Details on thermal modelling are given in Subsection E.

Alignment mismatch – For an alignment tuple a_k, we describe the spatial mismatch between desired alignment and actual layout as cost $c_{AMM}(a_k) = |rs(s_i, s_j)|$ (Subsection IV-B.3). Overall cost is then calculated as $c_{AMM} = \sum_{a_k \in A} c_{AMM}(a_k)$.

B. Optimization Phases

We consider two different phases for SA optimization; these phases support efficient solution-space exploration and layout optimization for 3D floorplanning with block alignment.

Phase I, "Fixed-Outline Fitting" – The cost function is defined as $c_{FOF} = c_{OL}$. Note that we do not perform alignment (Subsection IV-B.3) in this phase. The reason for initially focusing SA's search solely on the fixed-outline is simply that non-fitting layouts are a "knock-out", regardless of any achieved block alignment and layout optimization. The transition to phase II is made when the SA search triggers the first, fixed-outline fitting layout.

Phase II, "Alignment and Layout Optimization"– We compose the cost function as

$$\begin{array}{lcl} c_{ALO} & = & c_{OL} + (1 - \alpha) \times \sum_{c' \in C'} c' \\ C' & : & \{\beta(c_{WL}/c_{WL_{init}}),\ \gamma(c_{TSVs}/c_{TSVs_{init}}), \\ & & \delta(c_T/c_{T_{init}}),\ \epsilon(c_{AMM}/c_{AMM_{init}})\} \end{array}$$

with $\beta + \gamma + \delta + \epsilon \leq 1$. Note that we memorize *initial cost terms* like $c_{WL_{init}}$ during transition to phase II, i.e., we derive them from the first valid solution. Furthermore, note that we consider c_{OL} as essential term in this phase as well; based on our experiments, the SA search for comprehensively optimized layouts still depends on outline fitting/optimization.

C. Layout Operations

We consider the following set of layout operations to support the SA heuristic in effective exploration of Corblivar's solution space: swapping blocks within or across dies (CBL sequences), swapping or moving whole CBL tuples within or across CBL sequences, switching a block's insertion direction, switching a block's T-junctions, rotating hard blocks, and guided shaping of soft blocks as proposed in [19].

For optimization phase I, operations and blocks / CBL tuples are selected randomly. In phase II, blocks related with failed alignment requests are particularly selected. These blocks are swapped with adjacent blocks such that $|rs(s_i, s_j)|$ is reduced, i.e., such that the alignment is more likely to be fulfilled.

D. Cooling Schedule

As indicated earlier, we require an adaptive cooling schedule for improved efficiency of solution-space exploration. Our schedule is capable of (i) guiding the SA search within the global phases and (ii) escaping local minima. The schedule is composed of three different phases, explained below. Note that i labels the current step of i_{max} total temperature steps.

Phase "Adaptive Cooling" – We apply this cooling phase during SA phase I, which is aiming for fixed-outline fitting.

$$T_{i+1} = \left(cf_1 + \frac{i-1}{i_{max}-1} \times (cf_2 - cf_1)\right) \times T_i$$

The cooling rate slows down (given that $cf_1 < cf_2 < 1.0$); our intention here is to achieve initially fast cooling for the global scope, followed by slower cooling in a confined, "local" solution space.

Phase "Reheating and Freezing" – This is applied for SA phase II, i.e., after a fitting layout was found in step i_{first}.

$$T_{i+1} = \left(1 - \frac{i-i_{first}}{i_{max}-i_{first}}\right) \times cf_3 \times T_i$$

The cooling rate increases steadily; however, setting $cf_3 > 1.0$ results in an initial reheating for $i \geq i_{first}$. This way, the SA search has an increased flexibility for accepting high-cost solutions in this "interesting solution-space region" covering the first fitting layout. According to experiments, this limits the risk for being subsequently trapped in solution-space minima.

Phase "Brief Reheating" – This phase enables a somewhat "autonomous" and robust cooling schedule.

$$T_{i+1} = cf_4 \times T_i,\ cf_4 > 1.0$$

It is applied in alternation with the phase "reheating and freezing" for *individual* temperature steps during SA phase II. Such brief reheating helps the SA search to escape local minima; it is applied when we observe $\sigma(c_{ALO}) \sim 0$ during previous k steps, that is when the search reached a local "cost plateau". This technique is inspired by Chen and Chang's study [19]; their approach, however, proposes reheating solely at one particular temperature step, which we believe is not as effective as our cost-controlled reheating.

E. Fast Evaluation of Thermal Distribution

For fast yet accurate (steady-state) temperature analysis, we extend the work of Park et al. on *power blurring* [20]. Instead of using computationally intensive finite-differences or finite-elements analysis (FEA), power blurring is based on simple matrix convolution of thermal impulse responses and power-density distributions. (Park et al. reveal promising results when comparing to *ANSYS FEA* runs; they achieve maximal errors of less than 2% with computation speedups of $\sim 60\times$.)

For improved efficiency and to provide an integrated floorplanning tool [18], we refrain from time-consuming FEA runs for retrieving thermal masks [20]. Instead, we model the masks' underlying thermal impulse responses as *2D gauss functions* $g(x, y, w, s) = w \exp\left(-\frac{1}{s}x^2\right) \exp\left(-\frac{1}{s}y^2\right)$ with w as amplitude-scaling factor and s as lateral-spreading factor. To obtain the whole set of required masks [20], we need some scaling measure for g. We thus adapt w for each die d_i's mask such that $w_i = w/(i^{w_s})$, where $\max(i)$ represents the uppermost die next to the heatsink and w_s denotes a scaling parameter. For actual parametrization of w, w_s and s, we determine for each different 3D-IC setup (w.r.t. die count and dimensions) (i) an exemplary thermal distribution using a 3D-IC extension of *HotSpot* [21], a state-of-the-art academic thermal analyzer, and (ii) a best fit for above parameters based on a local search using *HotSpot*'s solution as reference model.

VI. Experimental Results

We conduct several experiments described below to validate Corblivar's capabilities. Relevant configuration details are given in Subsection A; results are discussed in Subsection B.

Structural planning of interconnects – We consider a set of several interconnects running both within and across dies; the (arbitrarily defined) set contains 10 width- and length-limited buses, each covering up to 5 blocks, along with 3 block pairs to be vertically aligned. We assume that each interconnect structure bundles 64 signals. For structural planning of these interconnects, in total 18 blocks have to be aligned simultaneously; related alignment tuples can be retrieved from [18]. Such scenario has not been considered in previous studies, thus we cannot meaningfully compare to other work.[3]

Regular and large-scale 3D floorplanning – To evaluate Corblivar's efficiency w.r.t. key 3D floorplanning objectives, we look into layout packing, wirelength and thermal optimization. (Note that we refrain from deriving alignment tuples for the considered benchmarks' nets. In other words, here we do not apply block alignment for interconnects planning and/or wirelength optimization.) We compare our work to relevant previous studies [22, 23]. Furthermore, we demonstrate Corblivar's scalability by utilizing the *IBM-HB+* benchmark suite [24]. To the best of our knowledge, this is the first time that these large-scale circuits are considered for 3D floorplanning.

A. Configuration

3D-IC configuration and benchmarks – We assume face-to-back stacking of two or three dies. Dies are $100\mu m$ thick; further properties are given in [18]. Terminal pins are only available on the lowermost die which is assumed to be connected to the package board. Practical (i.e., stackable) fixed outlines are ranging from $10mm \times 10mm$ up to $15mm \times 15mm$. We consider *GSRC* [25] and *IBM-HB+* [24] circuits. For reasonable utilization of die outlines, benchmarks are enlarged. In this context, power-density values are scaled down by factor 10. Also, results are referring to packed layouts where feasible, i.e., reduced outlines are reported. Deadspace utilization by $10\mu m \times 10\mu m$-sized TSVs was negligible in most cases, we thus refrain from optimizing and reporting TSV counts.

Setup – We conduct all experiments on a *Intel Core 2* system; reported runtimes are thus comparable. Corblivar and [23] are embedded in SA-based tools; best results are chosen from 5 up to 25 runs. Applied Corblivar parameters are retrievable from [18]. For *HotSpot*, default settings are applied [21].

[3] Previous studies on block alignment for 3D ICs have looked into differing scenarios. Nain and Chrzanowska-Jeske [5] propose techniques to split up and align (sub-)modules among adjacent dies with fixed (zero) offsets. They neglect to provide derived benchmarks containing split-up blocks, thus a comparison is hindered. Law et al. [6] consider a more flexible problem formulation; for vertical bus planning, they define sets of blocks for each die separately and require (at least) one block from each set to be vertically aligned with one block from the other sets. This simplified alignment problem is not compatible with our approach where we require all specified blocks to be aligned. Li et al. [7] indicate capabilities for block alignment but refrain from providing further details and related experimental results. Finally note that all aforementioned studies exclusively consider vertical alignment with fixed offsets.

B. Results

Structural planning of interconnects – We observe that the entire set of interconnects is successfully integrated, i.e., all related blocks can be simultaneously aligned (upper part of Table I). Compared to experiments where planning of interconnects is ignored (lower part of Table I), we expect and observe an increase of die outlines and deadspace—block alignment limits the flexibility of layout packing. More importantly, however, we observe notable wirelength increases in case of neglected interconnects planning; these overheads arise from routing detours for interconnects embedded in unaligned blocks. Finally, fixed die outlines were fulfilled in any case, i.e., the proposed SA optimization phases are effective.

An example for successful interconnects planning with corresponding block alignment is illustrated in Fig. 7.

Regular 3D floorplanning – Next, we discuss results on conducting floorplanning with applied layout packing and (equal) consideration of thermal and wirelength optimization (Table III). We observe that Corblivar is competitive with a force-directed tool [22] and superior to a SA-based tool [23]; both represent state-of-the-art academic works. In particular, we achieve comparable wirelengths and temperatures as [22] but with reduced die outlines and deadspace ratios. This indicates the efficiency of layout packing, which is most likely achieved by the proposed virtual CBL adaption. Comparing to [23], however, we note that Corblivar's layouts exhibit larger deadspace ratios and thus reduced packing densities. Nonetheless, we achieve reduced wirelengths in most cases. Also, the high packing density of [23] comes at a price; maximal temperatures are notably increased by tens of Kelvins compared to Corblivar. Thus, our tool effectively addresses the trade-off between packing density and maximal temperature. Furthermore, fixed outlines were fulfilled in these experiments as well.

As for our temperature analysis, we observe that it shows some local deviations compared to *HotSpot*-verification runs (Fig. 8). As indicated in [20], convolution-based thermal analysis particularly induces estimation errors at die boundaries. Thanks to our proposed mask parametrization, the actual thermal-distribution scale (i.e., the scale w.r.t. *HotSpot* runs) is matched nevertheless. For analysis during layout optimization, i.e., maximal-temperature estimation, our approach is thus applicable. It is also efficient due to fast computation; one run can be conducted in ∼20ms. (For comparison, one *HotSpot* run can take tens of seconds up to a few minutes.)

TABLE I
RESULTS ON ENLARGED GSRC BENCHMARKS FOR APPLIED INTERCONNECTS PLANNING, I.E., BLOCK ALIGNMENT (UPPER PART), COMPARED TO RESULTS FOR FLOORPLANNING WITHOUT INTERCONNECTS PLANNING (LOWER PART)

Metric	2 Dies			3 Dies		
	n100	*n200*	*n300*	*n100*	*n200*	*n300*
Wirelength ($cm \times 10^3$)	1.18	1.81	1.97	1.10	1.93	2.07
Die Outlines (cm^2)	1.14	1.14	1.14	0.73	0.84	0.91
Total Deadspace (%)	29.21	30.39	31.14	26.81	37.20	42.06
Runtime (s)	80	359	891	81	360	858
Wirelength ($cm \times 10^3$)*	1.83	2.60	2.53	1.34	2.59	2.76
Die Outlines (cm^2)	1.00	1.08	1.07	0.77	0.82	0.35
Total Deadspace (%)	18.82	27.04	26.39	29.81	36.00	32.19
Runtime (s)	59	304	726	59	304	734

*Estimated routing detours for (unaligned) interconnect structures are included.

TABLE II
RESULTS ON ENLARGED IBM-HB+ BENCHMARKS FOR LAYOUT PACKING AND WIRELENGTH OPTIMIZATION

Metric	2 Dies			3 Dies		
	ibm01	*ibm03*	*ibm07*	*ibm01*	*ibm03*	*ibm07*
Wirelength ($cm \times 10^3$)	4.77	7.29	1.77	4.67	7.39	1.67
Die Outlines (cm^2)	0.64	0.65	0.79	0.46	0.48	0.57
Total Deadspace (%)	17.24	19.26	18.59	24.97	27.86	24.88
Runtime (s)*	1195	3611	3081	1285	3792	3895

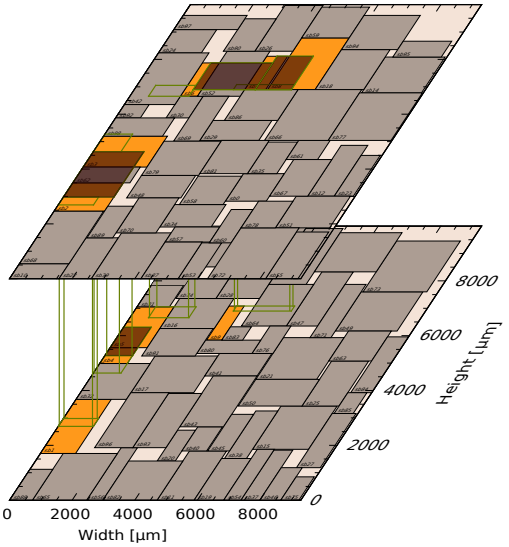


Fig. 7. Planned interconnect structures with corresponding block alignment, for enlarged benchmark *n100*. Vertical-bus sites are indicated by green, vertically extended rectangles; sites for 2D buses are colored as dark-brown. For interconnects planning aligned blocks are colored orange. For illustration purposes, we consider a reduced set of buses, covering blocks sb_1 to sb_9.

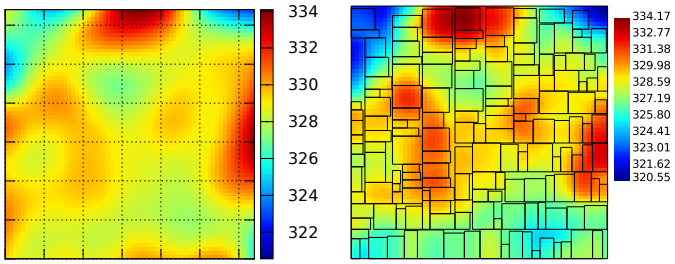


Fig. 8. Thermal maps of the critical die (furthest away from the heatsink), benchmark *n300*. The temperature scale of our fast analysis (a) matches with the scale obtained by running *HotSpot* (b); yet, local deviations are visible.

Large-scale 3D floorplanning – The IBM-HB+ suite does not include power information; we thus configured Corblivar only for wirelength and packing optimization (including successful consideration of fixed-outline constraints). Results on arbitrarily selected circuits are provided in Table II. We observe that total deadspace for these experiments is on average larger than for experiments on some GSRC circuits. This is expected and likely due to the fact that IBM-HB+ circuits contain up to ∼1,500 blocks where largest blocks are ∼33,000 times bigger than smallest ones; such designs are difficult to floorplan [26].

VII. SUMMARY

In this work, we extend 3D floorplanning towards structural planning of interconnects—an important yet inadequately addressed scenario for (future) massively interconnected 3D ICs. To tackle this omission of previous works, we promote block alignment. We initially discuss how 3D (inter-die) and 2D (intra-die) alignment can be applied for planning of diverse interconnects like vertical buses connecting (split-up) blocks on separate dies or classical 2D buses. We then introduce *Corb-*

TABLE III
COMPARATIVE RESULTS ON GSRC BENCHMARKS FOR LAYOUT PACKING WITH THERMAL AND WIRELENGTH OPTIMIZATION – BENCHMARKS ARE NOT ENLARGED FOR FAIR COMPARISON

Metric	Corblivar, 2 Dies				Corblivar, 2 Dies			Corblivar, 3 Dies				Corblivar, 3 Dies		
	n100	*n200*	*n300*	Avg	*ami33*	*xerox*	Avg	*n100*	*n200*	*n300*	Avg	*ami33*	*xerox*	Avg
Wirelength ($\mu m \times 10^5$)	3.70	6.57	9.07	6.45	2.02	13.89	7.96	4.24	7.19	10.28	7.24	2.06	16.36	9.21
Die Outlines ($\mu m^2 \times 10^5$)	1.01	0.99	1.61	1.20	10.38	143.15	76.77	0.75	0.68	1.08	0.84	8.98	117.48	63.23
Total Deadspace (%)	11.98	12.01	15.65	13.21	44.31	32.41	38.36	20.53	14.62	16.27	17.14	57.08	45.09	51.09
Max Temp [21] (°K)	313.81	314.53	315.95	314.76	309.14	353.85	331.49	355.62	363.94	363.35	360.97	333.17	416.61	374.89
Runtime (s)	108	286	548	314	50	14	32	154	380	704	413	71	22	47

Metric	[22], 2 Dies				[23], 2 Dies			[22], 3 Dies				[23], 3 Dies		
	n100	*n200*	*n300*	Avg	*ami33*	*xerox*	Avg	*n100*	*n200*	*n300*	Avg	*ami33*	*xerox*	Avg
Wirelength ($\mu m \times 10^5$)	3.65	6.18	9.53	6.45	1.81	17.14	9.48	4.59	7.17	10.61	7.46	2.22	21.86	12.04
Die Outlines ($\mu m^2 \times 10^5$)	1.19	1.21	2.15	1.52	9.65	125.17	67.41	0.97	0.82	1.48	1.09	7.54	88.93	48.24
Total Deadspace (%)	25.02	27.87	36.69	29.86	40.09	22.70	31.40	38.89	28.64	38.65	35.39	48.87	27.47	38.17
Max Temp [21] (°K)	313.31	313.74	314.63	313.89	336.36	366.48	351.42	348.55	360.60	361.35	356.83	384.39	482.16	433.28
Runtime (s)	439	446	526	470	193	47	120	266	497	574	446	193	48	121

livar, a 3D layout representation based on an extended corner block list with novel alignment tuples. To this end, we also develop effective techniques for block placement and alignment. We note that it is essential to synchronize alignment across the whole 3D IC—in particular, inter-die alignment requires to consider each die's layout in progress. Our techniques handle this appropriately for different scenarios of blocks to be aligned and/or to be placed. We embed Corblivar into an open-source, SA-based floorplanning tool; we also develop necessitated extensions like adaptive SA cooling and convolution-based fast thermal analysis. Experimental results on GSRC and large-scale IBM-HB+ benchmarks demonstrate Corblivar's applicability for structural planning of interconnects, i.e., block alignment, as well as its competitive performance for "classical" 3D floorplanning while considering fixed outlines, layout packing, thermal and wirelength optimization.

ACKNOWLEDGMENTS

The work of J. Knechtel was supported by the German Research Foundation (Project 1401/1). For the parametrization of the thermal model (Subsection V-E), Timm Amstein provided *Octave* scripts which are included in [18]. The authors thank Igor L. Markov for comments on drafts of this paper.

REFERENCES

[1] I. Loi *et al.*, "Characterization and implementation of fault-tolerant vertical links for 3-D networks-on-chip," *Trans. Comput.-Aided Des. Integr. Circuits Sys.*, vol. 30, no. 1, pp. 124–134, 2011.

[2] F. Li *et al.*, "Design and management of 3D chip multiprocessors using network-in-memory," in *Proc. Int. Symp. Comput. Archit.*, 2006, pp. 130–141.

[3] H. Xiang *et al.*, "Bus-driven floorplanning," in *Proc. Int. Conf. Comput.-Aided Des.*, 2003, pp. 66–73.

[4] J. H. Law and E. F. Young, "Multi-bend bus driven floorplanning," *Integration*, vol. 41, no. 2, pp. 306–316, 2008.

[5] R. Nain and M. Chrzanowska-Jeske, "Fast placement-aware 3-D floorplanning using vertical constraints on sequence pairs," *Trans. VLSI Syst.*, vol. 19, no. 9, pp. 1667–1680, 2011.

[6] J. H. Law *et al.*, "Block alignment in 3D floorplan using layered TCG," in *Proc. Great Lakes Symp. VLSI*, 2006, pp. 376–380.

[7] X. Li *et al.*, "A novel thermal optimization flow using incremental floorplanning for 3D ICs," in *Proc. Asia South Pacific Des. Autom. Conf.*, 2009, pp. 347–352.

[8] J. Knechtel *et al.*, "Assembling 2-D blocks into 3-D chips," *Trans. Comput.-Aided Des. Integr. Circuits Sys.*, vol. 31, no. 2, pp. 228–241, 2012.

[9] D. H. Kim *et al.*, "Block-level 3D IC design with through-silicon-via planning," in *Proc. Asia South Pacific Des. Autom. Conf.*, 2012, pp. 335–340.

[10] J. Knechtel *et al.*, "Multiobjective optimization of deadspace, a critical resource for 3D-IC integration," in *Proc. Int. Conf. Comput.-Aided Des.*, 2012, pp. 705–712.

[11] S. Panth *et al.*, "High-density integration of functional modules using monolithic 3D-IC technology," in *Proc. Asia South Pacific Des. Autom. Conf.*, 2013, pp. 681–686.

[12] H.-T. Chen *et al.*, "A new architecture for power network in 3D IC," in *Proc. Des. Autom. Test Europe*, 2011, pp. 1–6.

[13] K. Athikulwongse *et al.*, "Exploiting die-to-die thermal coupling in 3D IC placement," in *Proc. Des. Autom. Conf.*, 2012, pp. 741–746.

[14] S. K. Lim, "Folded 2-die block," Personal communication, March 2013.

[15] X. Hong *et al.*, "Corner block list: an effective and efficient topological representation of non-slicing floorplan," in *Proc. Int. Conf. Comput.-Aided Des.*, 2000, pp. 8–12.

[16] R. Fischbach *et al.*, "Investigating modern layout representations for improved 3D design automation," in *Proc. Great Lakes Symp. VLSI*, 2011, pp. 337–342.

[17] S. Chen *et al.*, "VLSI block placement with alignment constraints based on corner block list," in *Proc. Int. Symp. Circ. Sys.*, vol. 6, 2005, pp. 6222–6225.

[18] J. Knechtel. (2013) Corblivar floorplanning suite. [Online]: http://www.ifte.de/english/research/3d-design/index.html

[19] T.-C. Chen and Y.-W. Chang, "Modern floorplanning based on B*-Tree and fast simulated annealing," *Trans. Comput.-Aided Des. Integr. Circuits Sys.*, vol. 25, no. 4, pp. 637–650, 2006.

[20] J.-H. Park *et al.*, "Fast thermal analysis of vertically integrated circuits (3-D ICs) using power blurring method," in *Proc. ASME InterPACK*, 2009, pp. 701–707.

[21] A. Coskun *et al.* (2011, November) Hotspot 3D extension. [Online]: http://lava.cs.virginia.edu/HotSpot/links.htm

[22] P. Zhou *et al.*, "3D-STAF: scalable temperature and leakage aware floorplanning for three-dimensional integrated circuits," in *Proc. Int. Conf. Comput.-Aided Des.*, 2007, pp. 590–597.

[23] Y. Chen. (2010) 3DFP – thermal-aware floorplanner for three-dimensional ICs. [27]. [Online]: http://www.cse.psu.edu/~yxc236/3dfp/index.html

[24] A. N. Ng *et al.* (2006) IBM-HB+ benchmarks. [26]. [Online]: http://vlsicad.eecs.umich.edu/BK/ISPD06bench/

[25] (2000) GSRC benchmarks. [Online]: http://vlsicad.eecs.umich.edu/BK/GSRCbench/

[26] J. A. Roy *et al.*, "Solving modern mixed-size placement instances," *Integration, the VLSI Journal*, vol. 42, no. 2, pp. 262–275, 2009.

[27] W.-L. Hung *et al.*, "Interconnect and thermal-aware floorplanning for 3D microprocessors," in *Proc. Int. Symp. Quality Elec. Des.*, 2006, pp. 98–104.