Multiple Chip Planning for Chip-Interposer Codesign

Yuan-Kai Ho\(^1\) and Yao-Wen Chang\(^1,2,3\)

\(^1\)Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 106, Taiwan  
\(^2\)Department of Electrical Engineering, National Taiwan University, Taipei 106, Taiwan  
\(^3\)Research Center for Information Technology Innovation, Academia Sinica, Taipei 115, Taiwan

yuankai@eda.ee.ntu.edu.tw; ywchang@cc.ee.ntu.edu.tw

ABSTRACT

An interposer-based three-dimensional integrated circuit, which introduces a silicon interposer as an interface between chips and a package, is one of the most promising integration technologies for modern and next-generation circuit designs. Inter-chip connections can be routed on the interposer by chip-scale wires to enhance design quality. However, its design complexity increases dramatically due to the extra interposer interface. Consequently, it is desirable to simultaneously consider the co-design of the interposer and multiple chips mounted on it. This paper addresses the first work of chip-interposer codesign to place multiple chips on an interposer to reduce inter-chip wirelength. For this problem, we propose a new hierarchical B*-tree to simultaneously place multiple chips, macros, and I/O Buffers. An approach based on bipartite matching is then proposed to concurrently assign signals from I/O buffers to micro bumps. Experimental results show that our approach is effective and efficient for the codesign problem.

Categories and Subject Descriptors

B.7.2 [Integrated Circuits]: Design Aids

General Terms

Algorithms, Design

Keywords

Physical Design, 2.5D-IC, Interposer, Codesign

1. INTRODUCTION

As technology advances, interposer-based three-dimensional integrated circuits (interposer-based 3D ICs, and also known as 2.5D ICs) become one of the most promising solutions for enhancing system performance, decreasing power consumption, and supporting heterogeneous integration [14, 22]. Figure 1 shows the structure of an interposer-based 3D IC, which introduces a silicon interposer as an interface between chips and a package. Multiple chips can be mounted on the interposer, and inter-chip nets are routed on the redistribution layers (RDLs) of the interposer by chip-scale wires. In addition, different from stacked 3D ICs [11, 19] which use through-silicon vias (TSVs) as vertical interconnects among different layers, the interposer-based 3D ICs contain TSVs only in their interposer. As a result, the fabrication cost and design limitation/complexity with TSVs can be reduced, and many industrial products adopt the structure of the interposer-based 3D ICs, such as Xilinx Virtex-7 2000T [4, 12].

In a conventional design flow, chips are often designed independently, then placed on a silicon interposer, and finally routed with inter-chip connections on the RDLs of the interposer. This conventional flow might incur interposer-unfriendly micro bump assignments, and thus requires considerable extra efforts for inter-chip routing, causing a bottleneck of the time-to-market. To meet the micro bump assignments, furthermore, the inter-chip routing often incurs longer total wirelength and consumes more routing layers, dramatically degrading circuit performance and increasing manufacturing cost.

To improve the inter-chip routing quality, it is desirable to simultaneously consider a silicon interposer and multiple chips mounted on it. Figure 2 illustrates an example for the importance of chip-interposer codesign. In Figure 2(a), each of the four chips is designed independently and then placed on an interposer, resulting in a design with complex and hard-to-route inter-chip connections. The interposer, which mounts these chips, might need to be customized specifically for the design with more RDLs and thus significantly higher manufacturing cost. In contrast, Figure 2(b) shows another chip planning result considering the chip-interposer codesign. The inter-chip connections of this result are shorter and simpler than that in Figure 2(a). As a result, these inter-chip connections would
consume fewer layers and vias in a standard interposer and thus improve its signal quality and reduce the manufacturing cost.

Many previous works have addressed various codesign problems as follows: (1) chip-package codesign [5, 6, 13, 17, 18, 21, 23], (2) package-board codesign [8, 16], and (3) chip-package-board codesign [7, 15, 20]. However, no previous work is focused on silicon interposers, key components of interposer-based 3D ICs. For chip-package codesign, Fang and Chang [5] proposed a network-flow based routing algorithm to route free-assignment signals from block ports to bump balls. Xiong et al. [23] defined some constraints for chip-package codesign and proposed an effective multi-step algorithm to solve a constraint-driven I/O placement problem. In addition, some previous works focused on the signal skew between chips and packages [6, 13, 18, 21]. For package-board codesign, Fang et al. [8] used Delaunay triangulation and Voronoi diagrams to create triangular tile models and then route signal nets with the models. The work [16] presented a pin-out assignment method to consider the interactions between packages and boards. For the chip-package-board codesign problem, Park [20] proposed a die abstract model and a board-driven methodology to reduce the overall design cycle and cost. Lee and Chang [15] proposed a L-shaped codesign flow to route signals among boards, packages, and chips. To consider differential pairs, an integer linear programming (ILP) based routing algorithm was proposed in [7]. Although these previous works have addressed many codesign issues, the inter-chip connection optimization for chip-interposer codesign has not been considered.

In this paper, we address the chip-interposer codesign problem and propose an effective multiple chip planning algorithm to minimize inter-chip wirelength for interposer-based 3D IC designs. We first propose a new hierarchical B*-tree (HB*-tree, for short) to simultaneously place multiple chips, macros, and I/O Buffers. We also propose a cost function to evaluate an HB*-tree for simulated annealing (SA) based optimization. Finally, an approach based on bipartite matching [3] is presented to assign signals from I/O buffers to micro bumps. Experimental results show that our approach is effective and efficient for the chip-interposer codesign problem.

We summarize our contributions as follows.

- This paper presents the first work to address the problem of chip-interposer codesign. To the best of our knowledge, no previous codesign work considers silicon interposers, key components of novel interposer-based 3D ICs.

- A new HB*-tree is proposed to solve the addressed problem effectively in reasonable running time. The HB*-tree can be used to simultaneously place multiple chips, macros, and I/O Buffers.

- A bipartite-matching-based algorithm [3] is proposed to find the connections between I/O buffers and micro bumps.

The rest of this paper is organized as follows. Section 2 gives the formulation of the chip-interposer codesign problem and then reviews the B*-tree representation [1] on which our work is based. Section 3 proposes our planning algorithm for interposer-based 3D IC designs. Section 4 reports our experimental results, and Section 5 gives the conclusion.

2. PRELIMINARIES

In this section, we first formulate the chip-interposer codesign problem in Section 2.1. Then, we review the B*-tree representation [1] in Section 2.2 because our proposed algorithm is based on this representation.

2.1 Problem Formulation

We first give some notations used in this paper:

- \( C = \{c_1, c_2, \ldots, c_{|C|}\} \) is the set of chips mounted on a silicon interposer.
- \( M_i = \{m_{i1}, m_{i2}, \ldots, m_{ik_i}\} \) is the set of macros in the chip \( c_i \).
- \( B_i = \{b_{i1}, b_{i2}, \ldots, b_{ib_i}\} \) is the set of I/O buffers in the chip \( c_i \).
- \( U_i = \{u_{i1}, u_{i2}, \ldots, u_{iub_i}\} \) is the set of micro bumps in the chip \( c_i \).
- \( M = M_1 \cup M_2 \cup \ldots \cup M_{|C|} \) is the set of all macros.
- \( B = B_1 \cup B_2 \cup \ldots \cup B_{|C|} \) is the set of all I/O buffers.
- \( U = U_1 \cup U_2 \cup \ldots \cup U_{|C|} \) is the set of all micro bumps.

The predefined wire connections in a silicon-interposer-based design can be classified as follows: (1) intra-chip connections and (2) inter-chip connections. Intra-chip connections define the connections of macros in \( M_i \) inside a single chip \( C_i \), \( 1 \leq i \leq |C| \), while inter-chip connections define the wire connections among different chips through I/O buffers. Note that we need to connect these buffers to suitable micro bumps in our proposed algorithm to accomplish the inter-chip connections because these connections are routed on a silicon interposer. Figure 3 illustrates an instance of a silicon-interposer-based design. Since the chips in \( C \) are area-I/O flip-chip designs, the I/O buffers in \( B \) can be placed in the whole area of a chip.

The problem of multiple chip planning in an interposer-based design can be defined as follows:

- **Chip-Interposer Codesign Problem**: Given \( C, M, B, U \), and a silicon interposer, place macros and I/O buffers within corresponding chips without any overlaps, and find the locations of the chips on the interposer, so that no constraints are violated, and the total wirelength is minimized.
2.2 Review of B*-tree

Our proposed algorithm is based on the B*-tree representation presented in [1] to tackle the chip-interposer codelign problem. Therefore, we shall give a brief review of the B*-tree representation.

Given a compacted placement, where each module cannot move bottom and left anymore, we can construct a B*-tree to represent the compacted placement. A B*-tree is an ordered binary tree, and each node of the B*-tree represents a module of a compacted placement. The root of a B*-tree corresponds to the module on the bottom-left corner. In addition, a B*-tree has the geometric relationship between two nodes

\[ \text{node } n_i \text{ and node } n_j \text{ as follows: (1) if } n_j \text{ is the left child of the node } n_i, \text{ the corresponding module of } n_j \text{ is the lowest adjacent module on the right side of the corresponding module of } n_i, \text{ and (2) if } n_j \text{ is the right child of the node } n_i, \text{ the corresponding module of } n_j \text{ is the first module above the corresponding module of } n_i \text{ with the same x-coordinate.} \]

As a result, we can derive the \( x \)-coordinate of each module in a compacted placement by traversing the corresponding B*-tree.

Figure 4 gives an example of a compacted placement and its corresponding B*-tree [1]. For a node \( n_i \) and its corresponding module \( m_i \), the left child of the node \( n_i \) represents the lowest adjacent module on the right of the corresponding module \( m_i \); the right child of the node \( n_i \) represents the first module above the corresponding module \( m_i \) with the same \( x \)-coordinate.

To find feasible micro bumps and connect them to I/O buffers. See the following sections for the detailed algorithms.

3.3 HB*-tree Construction

Given a set \( C \) of chips, a set \( M \) of macros, and a set \( B \) of I/O buffers, we can construct the corresponding nodes in our hierarchical B*-tree (HB*-tree for short). The nodes of an HB*-tree can be categorized as follows:

- \( N_c = \{n_1^c, n_2^c, \ldots, n_{|N_c|}^c\} \) is the set of chip nodes, corresponding to the chips in \( C \).
- \( N_m = \{n_1^m, n_2^m, \ldots, n_{|N_m|}^m\} \) is the set of macro nodes, corresponding to the macros in \( M \).
- \( N_b = \{b_1^b, b_2^b, \ldots, b_{|N_b|}^b\} \) is the set of buffer nodes, corresponding to the I/O buffers in \( B \).
- \( N_m = N_{m_1} \cup N_{m_2} \cup \ldots \cup N_{m_{|C|}} \) is the set of all macro nodes.
- \( N_b = N_{b_1} \cup N_{b_2} \cup \ldots \cup N_{b_{|C|}} \) is the set of all buffer nodes.

To reduce the wirelength between I/O buffers and macros, we cluster a macro and its connected I/O buffers into a group. For this group, we introduce more notations as follows:
3.2 HB*-tree Perturbation and Packing

To perturb an HB*-tree in simulated annealing, we apply the following operations.

- **Op1**: Rotate a chip, a macro, an I/O buffer, or a macro-buffer group.
- **Op2**: Move a node of a subtree to another place of the same subtree.
- **Op3**: Swap two nodes within a subtree.

For Op1, we rotate a chip, a macro, an I/O buffer, or a macro-buffer group for a tree node. For Op2, we only allow a node of a subtree to be moved to another place of the subtree due to the special structure of an HB*-tree. For example, moving a chip node to the left child of another chip node is legal, but moving a macro node from one chip to another chip is illegal. Similarly, for Op3, we only allow swapping two nodes within one subtree.

We pack an HB*-tree from the bottom level to the top level. That is, we first pack each subtree in the third level and then pack that in the second level. Finally, each subtree in the first level is packed. The subtree packing is the same as the B*-tree packing by applying the depth first search (DFS) procedure.

### 3.3 Cost Function Evaluation

To evaluate the quality of a placement solution during simulated annealing, we define the cost function \( \Phi(P) \) of a placement \( P \) as follows:

\[
\Phi(P) = A + \alpha W_1 + \beta W_2 + \gamma \sum_{i=0}^{\left\lfloor \frac{|C|}{3} \right\rfloor} (R_i - R'_i)^2,
\]

where \( \alpha, \beta, \) and \( \gamma \) are user-specified weighting parameters, \( A \) is the total area of chips, \( W_1 \) is the total wirelength of inter-chip connections, \( W_2 \) is the total wirelength of intra-chip connections, \( R_0 \) is the current aspect ratio of the placement of all chips, \( R'_0 \) is the pre-defined aspect ratio of the placement of all chips on an interposer. \( R_i, 1 \leq i \leq \left\lfloor \frac{|C|}{3} \right\rfloor \), is the current aspect ratio of the chip \( i \), and \( R'_i, 1 \leq i \leq \left\lfloor \frac{|C|}{3} \right\rfloor \), is the pre-defined aspect ratio of the chip \( i \). Note that if the chip \( i \) is rotated, we adjust \( R'_i \) for the rotation.

To prevent a placement solution from out of the boundaries of chips, the aspect ratio penalty, \( \sum_{i=0}^{\left\lfloor \frac{|C|}{3} \right\rfloor} (R_i - R'_i)^2 \), is adopted in the cost function. As suggested in [2], we use the square of the aspect ratio difference and impose a huge weight for the aspect ratio penalty to guarantee that our placement solution can place all macros and I/O buffers into the corresponding chips, and also place all chips on an interposer. Then, we shall decrease the weight of the aspect ratio penalty to concentrate more on the wirelength optimization.

Since we have not established the connections between I/O buffers and micro bumps yet, we shall use the Manhattan distance between two I/O buffers as the wirelength of inter-chip connections in the cost function. We will assign signals from I/O buffers to micro bumps in Section 3.5.
3.4 Chip Separation

Since we derive a compacted placement of chips after the SA-based optimization, a partition-based approach is proposed to spread the chips in the whole feasible region of a silicon interposer. In our approach, we vertically (horizontally) partition the placement of chips into the left sub-placement and right sub-placement (top sub-placement and bottom sub-placement). Similarly, the feasible region of a silicon interposer can be partitioned in the same way. If a chip is in the right sub-placement (top sub-placement), we shift it with distance \( d \) along the x-axis (y-axis), where \( d \) is the distance between the sub-placement and the sub-region of the interposer. The partitioning is iteratively performed until the sub-placement size is sufficiently small, or each sub-placement has only one chip. Finally, we put the separated placement at the center of the feasible region of the silicon interposer.

Figure 8 illustrates our chip separation approach. As shown in Figure 8(a), we vertically partition the placement of chips and the feasible region of an interposer. The distance between the right sub-placement and the right sub-region is \( \Delta x \). Therefore, the chips in the right sub-placement are shifted with \( \Delta x \) along the x-axis as shown in Figure 8(b). Then, we horizontally partition each sub-placement and sub-region and derive the distances \( \Delta y_1 \) and \( \Delta y_2 \). The chips can further be separated as shown in Figure 8(c). Finally, we put the separated placement at the center of the feasible region as shown in Figure 8(d).

![Figure 8: An example of our chip separation approach. (a) The placement and the whole feasible region are partitioned vertically, and the chips in the right part of the placement are shifted with distance \( \Delta x \). (b) The placement and the feasible region are further partitioned horizontally, and the distances \( \Delta y_1 \) and \( \Delta y_2 \) are derived. (c) The top-left chip is shifted with distance \( \Delta y_1 \), and the top-right chip is shifted with distance \( \Delta y_2 \). (d) The separated placement in (c) is placed at the center of the feasible region.](image)

3.5 Micro Bump Assignment

The given inter-chip connections describe the wire connections among multiple chips through their I/O buffers. However, the multiple chips on a silicon interposer are connected each other by micro bumps. Therefore, we need to find the connections between I/O buffers and micro bumps. The micro bump assignment problem can be modeled as follows: Given a set \( B \) of I/O buffers and a set \( U \) of micro bumps, the objective is to assign signals from I/O buffers to micro bumps so that no constraint is violated, and the total wirelength is minimized.

For the assignment problem, we propose an algorithm based on bipartite matching [3] to establish the connections between I/O buffers and micro bumps. A bipartite graph \( G = (V, E) \) is first constructed, where \( V \) is \( B \cup U \), \( B \) is the set of I/O buffers, \( U \) is the set of micro bumps, and \( E \) is the set of edges between \( B \) and \( U \). For any pair, \( b_u^i \in B \) and \( u_b^j \in U \), we create an edge between the pair if they are in the same chip, that is, \( i = j \). In contrast, there is no edge between the pair if \( i \neq j \). We define \( c(b_u^i, u_b^j) \) to be the edge between \( b_u^i \) and \( u_b^j \). Figure 9 shows an example for constructing a bipartite graph. The I/O buffers and the micro bumps are in the two chips \( c_1 \) and \( c_2 \). There is an edge between an I/O buffer and a micro bump if the buffer and the bump are in the same chip. For example, there is an edge between \( b_2 \) and \( u_2 \) because they are in the same chip \( c_2 \), and there is no edge between \( b_2 \) and \( u_1 \) because they are in different chips.

The cost function \( \Psi \) of the edge \( e(b_u^i, u_b^j) \) is defined as follows:

\[
\Psi(e(b_u^i, u_b^j)) = \begin{cases} 
D(b_u^i, u_b^j) + 6D(u_b^j, b_2) & \text{if } b_2 \text{ connected with } b_u^i \\
D(b_u^i, u_b^2) & \text{otherwise,}
\end{cases}
\]

where \( \delta \) is a user-specified parameter, \( D(b_u^i, u_b^j) \) is the distance between the I/O buffer \( b_u^i \) and the micro bump \( u_b^j \), and \( D(u_b^j, b_2) \) is the distance between the bump \( u_b^j \) and the I/O buffer \( b_2 \). If the buffer \( b_u^i \) in chip \( i \) connects with the buffer \( b_2 \) in chip \( j \), we want to select a bump so that the sum of the distance between the buffer \( b_2 \) and the bump, and the distance between the bump and the buffer \( b_u^i \) is minimized. If the buffer \( b_u^i \) has no pre-defined connection with another buffer, we only consider the distance between the buffer \( b_2 \) and its connected micro bump.

After constructing the bipartite graph \( G \), we apply a bipartite matching algorithm to match the I/O buffers with the micro bumps based on the proposed cost function.

![Figure 9: Bipartite graph construction. An edge between an I/O buffer and a micro bump is introduced if the buffer and the bump are in the same chip.](image)

4. EXPERIMENTAL RESULTS

We implemented our algorithm in the C++ programming language and performed experiments on an Intel Xeon 2.93GHz Linux workstation with 48GB memory.

To validate the effectiveness of our approach, we tested our approach on six industrial testcases. We slightly modified these testcases to suit the interposer-based 3D IC structures. Table 1
shows the statistics of these testcases. Column “#Chips” represents the number of chips in a silicon interposer. Columns “#Macros” and “#Buffers” give the total number of macros and I/O buffers in all chips, respectively. The number of inter-chip connections is listed in Column “#Inter-chip nets”.

<table>
<thead>
<tr>
<th>Circuits</th>
<th>#Chips</th>
<th>#Macros</th>
<th>#Buffers</th>
<th>#Inter-chip nets</th>
</tr>
</thead>
<tbody>
<tr>
<td>case1</td>
<td>2</td>
<td>12</td>
<td>50</td>
<td>15</td>
</tr>
<tr>
<td>case2</td>
<td>2</td>
<td>24</td>
<td>336</td>
<td>55</td>
</tr>
<tr>
<td>case3</td>
<td>2</td>
<td>46</td>
<td>640</td>
<td>127</td>
</tr>
<tr>
<td>case4</td>
<td>2</td>
<td>56</td>
<td>768</td>
<td>179</td>
</tr>
<tr>
<td>case5</td>
<td>4</td>
<td>86</td>
<td>1194</td>
<td>304</td>
</tr>
<tr>
<td>case6</td>
<td>4</td>
<td>92</td>
<td>1280</td>
<td>514</td>
</tr>
</tbody>
</table>

We compared our approach with a conventional design flow because no existing work has addressed the chip-interposer code-sign problem. We say that the design flow is a single-chip-driven method, which optimizes a chip at a time without considering the other chips. For the single-chip-driven method, the macros and the I/O buffers of each chip are placed first, and then signals are sequentially assigned from I/O buffers to their closest micro bumps. Finally, all chips are put in a silicon interposer. Table 2 shows our experimental results. Column “Wirelength” lists the total wirelength of inter-chip nets and intra-chip nets; column “CPU times” gives the total running time of each testcase. In addition, since the single-chip-driven method is to individually place chips, we sum up the running time of each chip as the total running time of the method.

<table>
<thead>
<tr>
<th>Circuits</th>
<th>Wirelength</th>
<th>CPU times (s)</th>
<th>Wirelength</th>
<th>CPU times (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>case1</td>
<td>92080</td>
<td>6</td>
<td>84826</td>
<td>9</td>
</tr>
<tr>
<td>case2</td>
<td>513204</td>
<td>32</td>
<td>482508</td>
<td>43</td>
</tr>
<tr>
<td>case3</td>
<td>993020</td>
<td>172</td>
<td>855264</td>
<td>210</td>
</tr>
<tr>
<td>case4</td>
<td>1686745</td>
<td>216</td>
<td>1409120</td>
<td>295</td>
</tr>
<tr>
<td>case5</td>
<td>3969600</td>
<td>340</td>
<td>3149400</td>
<td>476</td>
</tr>
<tr>
<td>case6</td>
<td>6616723</td>
<td>358</td>
<td>5154129</td>
<td>542</td>
</tr>
<tr>
<td></td>
<td>1.17</td>
<td>0.72</td>
<td>1.00</td>
<td>1.50</td>
</tr>
</tbody>
</table>

Compared with the single-chip-driven method, our proposed algorithm can reduce the total wirelength for all testcases in reasonable running time. This is because our proposed algorithm can simultaneously place chips, macros, and I/O buffers of an interposer to consider the inter-chip connections. By observing the comparison, we find that we get higher improvements for larger cases with bigger chip sizes or larger chip numbers. The reason is that these larger cases have many possible locations of I/O buffers and complex inter-chip connections; considering the inter-chip connections in these cases can significantly improve the total wirelength. In addition to the inter-chip connections, we also consider the very complicated intra-chip connections of each chip in our algorithm, where the locations of buffers and macros might be restricted by the intra-chip connections.

5. CONCLUSIONS

In this paper, we have addressed a chip-interposer code-sign problem for interposer-based 3D IC designs. For this problem, we have proposed a multiple chip planning algorithm. To consider the interaction between chips and an interposer, we have designed a three-level hierarchical $B^*$-tree to simultaneously place multiple chips, macros, and I/O Buffers. We also have proposed a cost function to evaluate an HB$^*$-tree for SA-based optimization. Further, to assign signals from I/O buffers to micro bumps, we have proposed a bipartite-matching-based algorithm that can find the connections between I/O buffers and micro bumps. Experimental results have shown that our approach is effective and efficient for the chip-interposer code-sign problem.

6. REFERENCES