Modular Placement for Interposer based Multi-FPGA Systems

Fubing Mao[1], Wei Zhang[2], Bo Feng[3], Bingsheng He[1], Yuchun Ma[3]
[1]School of Computer Engineering, Nanyang Technological University, Singapore
[2]Department of Electronic and Computer Engineering, HKUST, Hong Kong
[3]Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China
{fmao001, bshe}@ntu.edu.sg, wei.zhang@ust.hk, {fengb15, myc}@tsinghua.edu.cn

ABSTRACT

Novel device with multiple FPGAs on-chip based on interposer interconnection has emerged to resolve the IOs limit and improve the inter-FPGA communication delay. However, new challenges arise for the placement on such architecture. Firstly, existing work does not consider the detailed models for the path wirelength and delay estimation for interposer, which may significantly affect the placement quality. Secondly, previous work is mostly based on traditional tile-based placement which is slow for the placement of large design on multiple FPGAs.

In this paper, we propose a new fast two-stage modular placement flow for interposer based multiple FPGAs aiming for delay optimization with the incorporation of a detailed interposer routing model for wirelength and delay estimation. Firstly, we adopt the force-directed method for its global property to get an efficient solution as a start point of the placement. Secondly, we adopt the simulated annealing (SA) for its efficiency and effectiveness in searching the refinement solution. In order to speed up the refinement, the hierarchical B*-tree (HB*-tree) is employed to enable a fast search and convergence. The experiments demonstrate that our flow can achieve an efficient solution in a comparable time. The proposed approach is scalable to different design size.

Keywords

Silicon Interposer; Modular Placement; Hierarchical B*-tree

1. INTRODUCTION AND MOTIVATION

Recently, interposer-based FPGAs are proposed to significantly improve the logic capacity, interconnection bandwidth and performance [9]. For instance, Xilinx Inc., a major FPGA vendor, firstly develops the largest interposer-based FPGA, the Virtex-7 XC7V2000T, which has 4 dies and 1.95 million logic elements [9]. A general two-and-a-half-dimensional (2.5D) FPGA with interposer is shown in the Fig. 1 [9], where four FPGA dies are placed vertically and are connected by the interposer. The interposer is an electric interface routing for connecting different components and can be used to spread connections, e.g., providing alternative routes for a

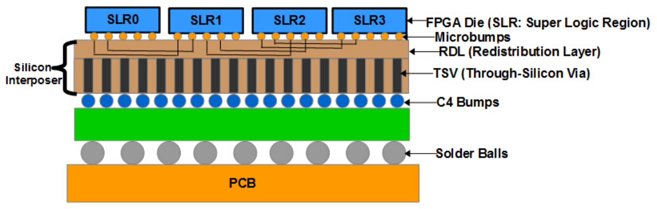

SLR0
SLR1
SLR2
SLR3
FPGA Die (SLR: Super Logic Region)
Microbumps
RDL (Redistribution Layer)
Silicon Interposer
TSV (Through-Silicon Via)
C4 Bumps
Solder Balls
PCB

Figure 1: A general 2.5D FPGA with interposer.

connection [9]. In order to connect between different dies, the signal needs to go through the internal track of the current die, microbump, and super long line (SLL) of interposer, and then into the other die.

FPGA placement is a very important step for synthesis in the modern FPGA [13] which significantly affects the implementation delay and the optimal placement is a NP-complete problem [7]. The increasing design complexity and logic capacity of the FPGA system, especially the multi-FPGA system, have further complicated the problem. In traditional design flow, fine-grained tile-based (CLB level) placement is performed to achieve the efficient solution. However, it usually requires a prohibitively long searching time from hours to days, especially for a large-scale design [16]. To accelerate the placement of large-scale design, module based placement has been employed for single FPGAs [2]. Here, module represents a rectangle region containing some logic resources and each module can be a reconfigurable region [26]. To our best knowledge, there is no existing flow to support the module based placement for multiple FPGAs, especially taking into consideration the new interconnect architecture through interposer.

In this work, we propose a new flow to address the placement for multi-FPGA system with interposer targeting delay optimization. Our approach consists of two steps. Firstly a force-directed placement is performed to obtain an efficiently initial solution. Secondly, the initial solution is represented by the Hierarchical B*-tree (HB*-tree) [5] and based on it, simulated annealing is performed to obtain the refined solution. A detailed routing model for interposer is built and integrated in the flow for addressing the interposer impact on the placement. The B*-tree representation is introduced to speed up the search and convergence of the solution [5]. The HB*-tree matches the multi-FPGA system well where the multi-FPGA can correspond to a B*-tree and each chip can correspond to a sub B*-tree to enable the swap of modules in different chips. In addition, we also exploited the impact of different width-height ratios of the modules for delay optimization.

Our results demonstrate that 1) Interposer impacts the placement result significantly and compared with the tradi-

GLSVLSI '16, May 18-20, 2016, Boston, MA, USA
© 2016 ACM. ISBN 978-1-4503-4274-2/16/05...$15.00
DOI: http://dx.doi.org/10.1145/2902961.2903025

tional model, our proposed model can reduce the wirelength, total delay, delay from nets crossing the interposer by 8%, 7%, 19%, respectively; 2) The ratios of the module impact the placement results. When considering the different ratios of modules, the wirelength, total delay and delay from nets crossing interposer are improved by 8%, 4%, 9% compared to the results considering modules with only one ratio; 3) Our proposed flow can speed up the placement and achieve higher quality compared to the state-of-the-art min-cut based flow derived from [11].

2. RELATED WORK

2.1 Modular Placement for 2D single FPGA

There was a large amount of tile-based placement for a single 2D FPGA. They could be based on heuristic approach [19] or analytical approach [6] [30]. However, as we discussed in the previous section, tile-based placement faced great challenge for placement of large-scale design. Thus module based placement and hierarchical placement have been proposed to address the challenge as well as support the partial reconfiguration [2] [3]. However, placement in multiple FPGAs is fundamentally different from placement in a single FPGA due to the impact of inter-chip delay and assignment of modules into different chips.

2.2 Placement for 2D Multi-chip and 3D FPGA

Traditional multiple FPGAs had a limited number of IOs for passing signals between different dies. Hence, cut-size optimization was the common goal to reduce the number of signals crossing chips as well as the long inter-chip delay. [22] proposed an algorithm to minimize net crossings for addressing system partitioning and considered the block assignment of multi-chip modules during the high level design of an application-specific IC. [12] developed a new performance driven partitioning algorithm to implement a large circuit in multiple FPGAs to minimize the cut size. [25] developed a method named MP2 to minimize the total number of terminals of all blocks during network partitioning. [20] presented a rectilinear partitioning algorithm to minimize the cut sizes of the partitions and handle timing specifications for traditional multiple FPGAs.

Recently, 3D chip emerged for achieving higher integration density and better performance. The 3D FPGA has multiple dies stacked vertically which utilized through-silicon vias (TSVs) as vertical connections [14]. The TSV density was closely related to the cost and hence, most of 3D FPGA placements target the minimization of TSVs which was similar to the cut-size optimization. [1] adopted the updated popular partitioner hMetis derived from [11] to divide the circuit into several parts and minimize the connections between dies. [23] adopted a TSV-aware partitioning algorithm for 3D FPGA to achieve higher performance for applications. [14] proposed a reconvergence-aware layer partition (RALP) algorithm for 3D FPGA design targeting performance optimization. However, they were targeting different routing models and all were tile-based flow rather than module-based flow.

2.3 Placement for 2.5D IC and FPGA with Interposer

Little work has addressed the interposer-based FPGA placement, although there was some work proposed for ASIC floorplanning using the 2.5D integration with interposer. In [10], the authors proposed the first work of chip-interposer codesign to place multiple chips on an interposer-based IC to reduce the inter-chip wirelength. They also proposed a hierarchical B*-tree to simultaneously place multiple chips and IO buffers. The work in [15] presented an enumeration-based algorithm and a network-flow-based algorithm to solve the multi-die floorplanning and signal assignment problem for 2.5D IC respectively. [21] adopted simulated annealing for die placement and I/O assignment in 2.5D IC. However, all the approaches have been focusing on the ASIC flow instead of the FPGA flow. Since FPGA had a different architecture, placement methods targeting FPGA were needed to be developed and the approaches used in specific IC design were not directly applicable to the FPGA architecture.

The work in [9] was the first work to consider the tile based placement in the multi-FPGA system with interposer. In comparison, our work employs a much more detailed interposer routing model and mainly focuses on the development of module based placement flow for delay optimization.

3. PROBLEM FORMULATION

In this work, we aim to develop the modular placement flow for the interposer-based 2.5D FPGAs. A wirelength estimation model for the interposer routing is developed for the placement flow. We describe the notations used in the placement problem below. $E = \{e_1, e_2, ..., e_m\}$ is the set of dies in the 2.5D FPGAs. $IO = \{io_1, io_2, ..., io_k\}$ is the set of available IOs in the 2.5D FPGAs. $B = \{m_1, m_2, ..., m_n\}$ is the set of modules to be placed in the 2.5D FPGAs.

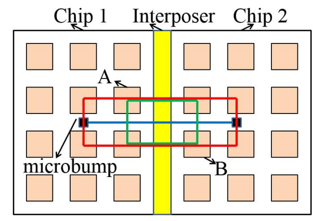
We define the problem of interposer based multi-FPGA placement as follows. Given a set of B with n rectangular modules $B = \{m_1, m_2, ..., m_n\}$, where each module has a width and height denoted by w_i, h_i, $1 \leq i \leq n$ respectively. The aspect ratio of module b_i is defined by w_i/h_i. Given a set of E with m dies $E = \{e_1, e_2, ..., e_m\}$, where each die has the same width and height denoted by EW, EH. Given a set of IO with k IOs $IO = \{io_1, io_2, ..., io_k\}$. A placement $p = \{(x_i, y_i)\}(1 \leq i \leq n)$ with modules (m_i) is an assignment of the rectangular modules such that the bottom-left corner coordinate of module m_i is assigned to (x_i, y_i), no two rectangular modules are overlapped, and the different dies are connected by interposer. A routing model described in 4.1 is used to estimate the wirelength and delay through interposer. We consider each function in the overall design as a rectangle module in this work, and the objective is to optimize the total delay of the mapped circuit as well as meet the IO and area constraints.

4. PROPOSED MODULAR PLACEMENT FLOW

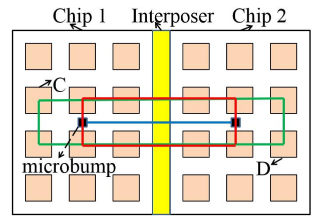
The complexity of the placement for a multi-FPGA system is much larger than that of a single FPGA. An efficient approach should consider both the runtime and placement quality. In order to optimize the placement as well as speed up the convergence, we take two stages to achieve the placement for the interposer based multi-FPGA system. Firstly, we adopt the force-directed method [30] for its global property to obtain a high-quality initial placement and then partition modules into different dies. Secondly, we adopt the simulated annealing (SA) [19] for its efficiency and effectiveness in finding the approximately optimal solution. In order to accelerate the refinement, we introduce the hierarchical B*-tree (HB*-tree) [10] to enable a fast search [5] and convergence. Moreover, a wirelength estimation model for the interposer is developed to allow a more accurate evaluation of the interposer impact on the placement flow.

4.1 Routing Model for Interposer

Based on the study of Xilinx commercial FPGA with interposer, e.g., Virtex-7 XC7V2000T [9], we know that the logic



(a) Original bounding box smaller than SLL length.



(b) Original bounding box larger than SLL length.

Figure 2: Difference of wirelength estimation when using interposer compared traditional bounding box calculation.

block in one FPGA die connects to another logic block in a different FPGA die in the following way. Firstly, the logic block in one FPGA die connects to the track (length-12) and the track connects to the corresponding microbump. Next, the microbump links to another microbump of the different FPGA die through the super long line (SLL) in the interposer [27]. Then, the signal goes through microbump and corresponding track and finally connects to the logic block in a different FPGA die. The SLL wires for connecting the adjacent FPGA dies through interposer have the same length as the chip length according to the Xilinx interposer-based FPGA, Virtex-7 XC7V2000T [9] [28]. To extend the model for general multiple FPGAs with interposer interconnection, we assume that any two adjacent chips can connect to each other directly through interposer and the SLL wire for connecting the adjacent FPGA dies through interposer have the same length as the die length, and the non-adjacent chips are connected by the intermediate chips.

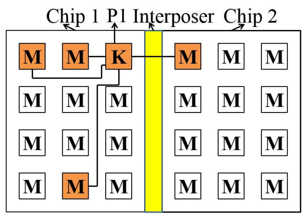
We use an example as shown in Fig. 2 to illustrate the difference of wirelength estimation when considering the interposer compared to traditional bounding box calculation. We assume that the CLB A in the FPGA die one is connected to the CLB B in FPGA die two. For the case in Fig. 2(a), we can not use the green bounding box for wirelength estimation as in the traditional model, since there is routing restriction that SLL only connects to the local tracks at its ending points. Instead, we consider the SLL length and use the red bounding box for wirelength estimation which is more accurate to model the interposer. For the case in Fig. 2(b) where the bounding box is larger than the SLL length, the traditional way of bounding box calculation can be applied.

4.2 Force-directed Placement

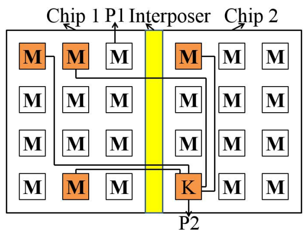
We adopt the force-directed approach [30] since it is a scalable approach for large designs and able to avoid being trapped in local minimum solution. It is also suitable for optimizing average delay of the mapped circuit. We adopt the force-directed algorithm which is similar to the [30]. However, we consider the interposer in the following two forces used in the force equation [30]. The basic definition can be found in [30] and we do not describe it due to the page limitation.

4.2.1 *Force Computing*

We define two types of force for balancing the placement result: Filling Force and Attraction Force [30]. The two forces are combined together using linear weighted function to form the d_x and d_y which is described in [30]. We adopt the approach (Barnes-Hut quad-tree for n-body force calculation) described in [4] to compute the filling force due to its accuracy, $O(nlog^n)$ complexity and high-quality of force calculation. To tradeoff the computation time and accuracy, in our work we model a quad-tree with at most ten levels for



(a) Original position of Module K.



(b) New position of Module K.

Figure 3: Example for attraction force considering interposer.

the placement region and recursively divide the placement region into grids with the termination condition that the bin size is not less than $(w_{min} + h_{min})/4$ [17], where w_{min}, h_{min} are the minimum width and height of all the modules.

Filling Force (f_x^F, f_y^F): it is used to remove overlap between modules and distribute the modules evenly over the 2D placement area. It pushes modules away from the regions with high density and pull the modules towards regions with low density in the 2D space (filling force direction). We define the density as the total module area that covers the bin. The filling force of each bin is equal to the bin density of each bin. The filling force of each module is equal to the sum of the filling force of all the bins that the module covers. We add an additional force for the modules which lie on the partition boundary to prohibit them from crossing the interposer and ease the final partition stage, since if modules are placed in a partition boundary, it is difficult to choose which die they belongs to.

Attraction Force (f_x^A, f_y^A): it is used to reflect the relation between the old coordinate (start point of the force) and new coordinate (end point of the force) of the module. The two different positions of the module form an attraction force. Since the delay in the interposer is larger than the wire delay with the same length in the single die, we consider the interposer in the attraction force in the following way. If the two positions of the module cross the interposer, we add an additional force where the force can be positive or negative. If the number of nets crossing interposer increases after moving the module to a new position, we will add a negative force, e.g., decrease the attraction force to avoid delay increase and vice versa. Fig. 3 shows an example for considering the interposer in the attraction force. We assume that Chip (die) 1 and Chip (die) 2 are two FPGA dies connected by interposer and 'M' represents a module. Currently we consider a module labelled with 'K' and the modules with color represent that they are used. P1 and P2 are the original position and new position for the module K, respectively. Fig. 3(a) shows one net crossing the interposer and Fig. 3(b) shows three nets crossing the interposer. Hence, we decrease the attraction force for moving K from P1 to P2 which may reduce the delay. Considering interposer in the force and cost function Eq. 1 of Section 4.3.4 can help to reduce the number of wires crossing the interposer.

4.2.2 *Partition and Legalization*

In each iteration of force-directed placement, we consider the interposer in the filling force and attraction force described above since the delay in the interposer is larger than the wire delay with the same length in a single die. Through considering interposer in the filling force it eases the partition stage. We divide the force-directed result into several parts according to their coordinates and chip boundary and each part represents a chip (die). After we assign the modules to

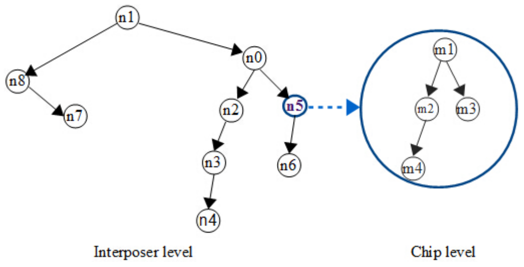


Figure 4: An Hierarchical B*-tree. In the interposer level, it has 9 chips. Chip n5 has 4 modules in the chip level.

the chips, we legalize (remove overlap) them using sequence pair [18] which is a floorplan representation and powerful for maintaining the relative relation of modules.

4.2.3 *Aspect Ratio Selection of Modules*

Since ratio impacts the circuit delay, we consider it in our flow. For each iteration of force calculation, we pick one of the ratio from the ratio list of the module and update its width and height. Thus it may impact the density of the bin that the modules cover. Through this way the filling force may change and finally it impacts the placement results.

4.3 Hierarchical B*-tree Based Module Placement

4.3.1 *Hierarchical B*-tree*

B*-tree is an ordered binary tree structure proposed in modern floorplanning of ASIC designs [5]. Compared to other data structures, B*-Tree provides faster search and area estimation, convenient handling of constraints, linear time transformation between the tree and placement [5]. B*-tree representation has used for the single FPGA placement [5]. To support the multi-FPGA placement, we adopt the hierarchical B*-tree (HB*-tree) since it matches the multi-FPGA system well in which the multi-FPGA corresponds to a B*-tree and each chip corresponds to a sub B*-tree. Moreover, HB*-tree based placement can enable a fast search [5] and convergence of the solution.

The hierarchical B*-tree consists of two levels: interposer level and chip level. At interposer level, each chip is a node of a B*-tree and the nodes determine the relative position of each chip. At chip level, each module is a node of sub B*-tree and the nodes in this level determine the placement of modules in each chip. Fig. 4 shows a structure of HB*-tree for illustration. At the interposer level, it has 9 chips and each chip is a node represented by n_i, $0 \le i \le 8$ respectively. At chip level, chip n_5 (one of the nodes in interposer level), contains 4 modules, and each module is a node represented by m_j, $1 \le j \le 4$ respectively.

4.3.2 *Hierarchical B*-tree Operations*

To explore the solution space, we apply the following operations.

At the interposer level, each chip is a node of the B*-tree and the following operations are performed on the nodes. **OP1:** Swap chips. **OP2:** Move a chip to another places.

At the chip level, each module is a node of the sub B*-tree and the following operations are performed on the nodes. **OP3:** Load module with shape of 1/(aspect ratio). **OP4:** Move module to an empty space. **OP5:** Swap two modules. **OP6:** Load a different aspect ratio of the module.

The following operations at the interposer level and chip level are performed at the same time. **OP7:** Move a module in a chip to another chip. **OP8:** Swap two modules that are in different chips.

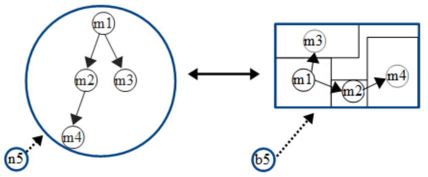


Figure 5: Chip-level packing: a sub-B*-tree of the chip n5 and its corresponding placement.

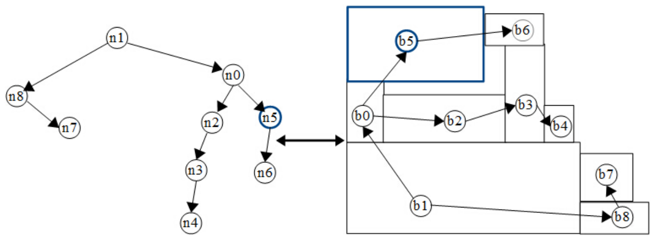


Figure 6: Interposer-level packing: a sub-B*-tree for chips and its corresponding placement

Here we use 'swap chips' (OP1) since all chips are the same and each node represent a chip in a tree of the interposer level. Swapping nodes in the tree is similar to swap chips.

4.3.3 *The Packing of the Hierarchical B*-tree*

We use two steps to pack the HB*-tree to a placement. Initially, we pack all the sub-B*-tree of the chip level one by one and get all the placement results for all the chips. Next, we pack the chip of interposer-level B*-tree to get a placement of the whole design. Finally, we map the placement of whole design using our proposed approach to the corresponding architecture.

A general example of a chip-level and interposer-level packing is shown in Fig. 5 and Fig. 6. Fig. 5 shows the sub-B*-tree of the chip $n5$ and its corresponding placement result. Fig. 6 shows that after packing the sub-B*-tree of each chip, we can pack the B*-tree at the interposer level, and each node of the B*-tree at the interposer level corresponds to the placement result of the chip. In our work, the dies are the same (same size: width and height).

4.3.4 *Cost Metric in HB*-tree based Module Placement*

To simplify the operation, we combine OP.3 and OP.6 together. We use τ which is area × delay of a shape of a module to evaluate the module generated by running VPR on the corresponding verilog file. The internal block information has already predefined, since VPR can achieve an efficient placement for the module. Simulated annealing is performed together with HB*-Tree operation to explore the solution space and the approach for choosing the temperature is the same as the [5]. In each iteration, the operations are randomly selected. Different weights are given to the operations to control the probability of choosing each operation. After each operation, the cost function [24] is calculated as shown in Eq. 1. Here Wirelength and Delay represent the current total wirelength and total delay containing all nets of the design respectively. $Wirelength^*$ and $Delay^*$ represent the average wirelength and delay of the design respectively.

$$Cost_{system} = \alpha(\frac{Wirelength}{Wirelength^*}) + (1-\alpha)\frac{Delay}{Delay^*} \qquad (1)$$

Table 1: Benchmarks From VPR Suite

Casename	Modules	Tiles	Casename	Modules	Tiles
bgm128	128	33160	bgm224	224	58030
bgm320	320	82900	spree121	121	2453
spree209	209	4237	spree308	308	6244
stereovision2112	112	60944	stereovision2210	210	40010
stereovision2308	308	180488	stereovision1105	105	20005
stereovision1210	210	40010	stereovision1315	315	60015

We divide the result by the average wirelength and delay to normalize the results. The α is a parameter to bias the cost function to wirelength or delay. If the net crosses the interposer, we will use our wirelength estimation model to calculate the wirelength and also consider the delay in the interposer described in detail in the Section 4.1. Thus the delay of each net includes the module delay and wirelength delay generated by Elmore delay [29]. We use the popular linear weighted function which is similar to [8] to bias the wirelength and total delay.

5. EXPERIMENTAL RESULTS

To evaluate the performance of our proposed flow, we first create benchmarks based on the VPR benchmark suite [19] by decomposing each of the verilog file into multiple verilog files according to subfunctions, which are known as modules. Each module with one aspect ratio is obtained by running VPR on each verilog file. Information of the cases are shown in Table 1. The number of modules ranges from 110 to 320 for the current case set. We also show the number of tiles contained to compare the difference in the size among these cases. The absolute tile values are for reference only since it varies in different architectures. In the experiment, each module has 9 ratios 0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.6, 2.5, 5.0 based on our evaluating and we set α to be 0.4. The multi-FPGA system is assumed to contain 6 chips (3 columns $\times$ 2 rows). All the experiments were performed on a IBM server x3650 with Intel Xeon(R) CPU and 42 GB DDR2 RAM.

5.1 Impact of Interposer

In the first experiment, we evaluate the interposer impact using proposed routing model to the force-directed placement for the initial placement and to the whole flow for the finally refined placement. In previous work, the constant delay, e.g., $1ns$ has been used for interposer [9]. Here we compare our results with the constant delay model used in [9] for interposer in our flow to show the impact of the detailed routing model. The results are listed in Table 2 and Table 3, respectively. In the table, **WL**, **Delay**, **CrossD**, **Time** denote the **total wirelength (tile)**, **total delay (ns)**, **the delay generated by the nets crossing the interposer (ns)** and the **runtime (s)**, respectively. The results in Table 2 show that our proposed model does not impact the force-directed placement much. The wirelength, total delay and delay from nets crossing the interposer are reduced by **1%**, **1%**, **5%** respectively, for the initial placement step. The time increases by 4%. However, the proposed model shows great impact on the final placement results for our flow. Table 3 shows that our proposed model reduces the wirelength, total delay, delay from nets crossing the interposer and runtime by **8%**, **7%**, **19%**, **2%** respectively. Hence these results demonstrate that a detailed routing model for interposer can help to improve the placement quality.

5.2 Impact of Different Module Ratios

We compare the results between modules with only one ratio and the modules with different ratios in the placement flow to show the impact of module ratios. Similarly, we evaluate the impact of different module ratios on the force-directed placement and the whole flow which usually uses one ratio in the previous work. The corresponding results are listed in Table 4 and Table 5. Table 4 shows that when considering different ratios of modules in force-directed placement, the wirelength, total delay and delay from nets crossing interposer are significantly improved by **18%**, **11%**, **12%** respectively with an extra 7% runtime. The results in Table 5 indicate that when considering different ratios of modules in the whole flow, the wirelength, total delay and delay from nets crossing interposer are improved by **8%**, **4%**, **9%** respectively and the runtime increase 46%. We can see that considering different module ratios in the force-directed placement can significantly optimize the total delay of the circuits and improve the placement quality. However, in the HB*-tree based solution refinement, the benefits from module ratios is reduced. It is because that when considering multiple module ratios in the SA, the design space is enlarged significantly and within a comparable time, only suboptimal placement can be reached by SA. While since force-directed placement is based on analytical solution, considering multiple module ratios will not increase its running time much.

Table 2: Comparison between Constant Delay Model used in [9] and Our Proposed Model in Enhanced Force-Directed Placement

Casename	Traditional Model used in [9]				Our Proposed Model			
	WL	Delay	CrossD	Time	WL	Delay	CrossD	Time
bgm128	22624	2140.28	145	9.49	21118	2052.59	104	10.63
bgm224	60676.5	4409.36	264	14.87	66748	4618.57	285	15.96
bgm320	110199.5	7072.01	421	45.26	105856.5	6936.38	420	48.67
spree121	10554.5	835	239	8.33	10416.5	827.73	236	8.57
spree209	23372	1575.85	387	39.15	24506.5	1617.02	393	39.74
spree308	48270	2810.62	630	132.3	48110	2775.66	600	135.97
stereovision1105	19941.5	1096.39	176	3.63	20353.5	1085.16	152	3.86
stereovision1210	59339	2892.92	449	16.09	54163	2672.46	389	16.7
stereovision1315	121681	5198.73	520	50.96	120186.5	5177.4	545	51.49
stereovision2112	61572	3035.49	296	4.86	62939	3042.87	261	4.83
stereovision2210	193964	8148.55	578	22.02	173945.5	7494.98	545	22.68
stereovision2308	330379.5	13498.35	972	68.55	342486.5	13877.66	976	69.94
Average	**1**	**1**	**1**	**1**	**0.99**	**0.99**	**0.95**	**1.04**

5.3 Speed Up of the Placement Flow

We compare our flow with the traditional flow (min-cut + SA) (module-level) [14] [11]. We use hmetis [11] for partitioning for its high quality and fast speed. Hmetis [11] is a software package for partitioning large hypergraphs and is based on min-cut. Its goal is to minimize the total cut weight. The weights of hyperedges and vertices are two parameters in Hmetis. Here the hyperedges and vertices are set to the same weight. The results are shown in the Table 6 and it shows that our approach improves the wirelength, total delay and the delay from nets crossing interposer by **21%**, **14%**, **17%** respectively in a shorter time (improve by **15%**). The benefits are achieved from the following two parts. On one hand, min-cut is optimized to minimize the cut size. Although it also helps on reduction of the delay, our force-directed placement can achieve similar or better initial placement due to its optimization for delay and consideration of the detailed interposer model. On the other hand, HB*-tree based SA can search and converge much faster than SA which enables our flow to find a more efficient solution than SA within the same time.

6. CONCLUSIONS

Our work proposes a module based mapping flow to address the placement for multi-FPGA systems with interposer. A routing model is developed to more accurately model the wirelength and delay of the interposer. The proposed mapping flow contains two stages. Firstly, we adopt the force-directed method to get a good-quality initial solution as a start point of the refinement step. Secondly, we adopt the

Table 3: Comparison between Constant Delay Model used in [9] and Our Proposed Model in Our Proposed Flow

Casename	Traditional Model used in [9]				Our Proposed Model			
	WL	Delay	CrossD	Time	WL	Delay	CrossD	Time
bgm128	14493	1849.22	106	26.24	11962	1722.75	58	28.27
bgm224	48274	4025.88	265	62.17	41004	3752.51	217	63.64
bgm320	77255	6028.74	399	144.58	77292	6022.88	392	126.04
spree121	11605	945.57	317	52.46	9706	775.7	206	53.4
spree209	28582	1935.36	585	152.58	25840	1702.35	437	158.78
spree308	57323	3391.27	930	344.77	52813	3083.46	762	331.17
stereovision1105	16639	990.01	172	64.93	16051	945.79	146	62.73
stereovision1210	63454	3116.48	545	207.44	56506	2772.1	416	204.56
stereovision1315	151125	6390.49	799	379.34	136079	5819.06	694	362.14
stereovision2112	37709	2215.74	216	58.3	33657	2017.12	143	58.59
stereovision2210	133026	6239.48	558	196.85	140514	6458.6	545	189.62
stereovision2308	236712	10409.65	787	351.94	247117	10784.21	839	313.57
Average	**1**	**1**	**1**	**1**	**0.92**	**0.93**	**0.81**	**0.98**

Table 4: Comparison between Modules with only one ratio and different ratios in Enhanced Force-directed Placement

Casename	Modules with Only One Ratio				Modules with Multiple Ratios			
	WL	Delay	CrossD	Time	WL	Delay	CrossD	Time
bgm128	21118	2052.59	104	10.63	17143.5	1933.66	97	12.41
bgm224	66748	4618.57	285	15.96	43157	3798.16	174	27.21
bgm320	105856.5	6936.38	420	48.67	72906.5	5847.29	324	74.55
spree121	10416.5	827.73	236	8.57	10399.5	832.27	233	8.67
spree209	24506.5	1617.02	393	39.74	24960	1662.51	413	40.07
spree308	48110	2775.66	600	135.97	48810	2852.7	640	134.77
stereovision1105	20353.5	1085.16	152	3.86	17306.5	991.03	141	4.05
stereovision1210	54163	2672.46	389	16.7	52014	2594.46	359	16.74
stereovision1315	120186.5	5177.4	545	51.49	111136	4841.78	460	52.42
stereovision2112	62939	3042.87	261	4.83	42946	2419.61	241	3.06
stereovision2210	173945.5	7494.98	545	22.68	116621.5	5649.92	446	16.04
stereovision2308	342486.5	13877.66	976	69.94	216465.5	9700.96	658	73.41
Average	**1**	**1**	**1**	**1**	**0.82**	**0.89**	**0.88**	**1.07**

simulated annealing (SA) to efficiently search the solution space and find the approximately optimal placement solution. In order to speed up the refinement of placement in SA, we employ the HB*-tree representation to enable a fast search and convergence. The mapping flow supports the modules with different ratios which impact the total delay of the circuit to achieve the delay optimization. Our proposed approach gets efficient result in comparable time and is able to scale for large design.

7. ACKNOWLEDGEMENT

This work is partly supported by a MoE AcRF Tier 2 grant (MOE2012-T2-1-126) in Singapore and by Grant R9336 from Hong Kong SAR.

8. REFERENCES

[1] C. Ababei. Tpr: Three-d place and route for fpgas. In *FPL*. 2004.

[2] S. Areibi and etc. Hierarchical fpga placement. *CJECE*, 2007.

[3] P. Banerjee and etc. Floorplanning for Partially Reconfigurable FPGAs. *TCAD*, 2011.

[4] J. Barnes. and P. Hut. A hierarchical $o(nlog^n)$ force calculattion algorithm. *Nature*, 1986.

Table 5: Comparison between Modules with only one ratio and different ratios in Our Proposed Flow

Casename	Modules with Only One Ratio				Modules with Multiple Ratios			
	WL	Delay	CrossD	Time	WL	Delay	CrossD	Time
bgm128	13015	1773.4	76	32.1	11205	1713.57	61	42.35
bgm224	47655	3994.69	253	69.22	39079	3686.74	189	101.59
bgm320	80676	6092.79	357	126.96	70302	5792.55	350	216.36
spree121	9112	739.29	188	56.87	8368	704.29	168	79.49
spree209	26101	1724.45	451	160.23	27454	1788.82	462	252.81
spree308	57054	3286.93	834	368.82	52148	3022.18	706	555.97
stereovision1105	14872	891.24	128	61.38	14386	892.49	133	98.11
stereovision1210	59430	2893.74	447	225.12	47977	2421.32	311	342.94
stereovision1315	130223	5576.53	633	440.75	122744	5379.63	638	581.6
stereovision2112	39531	2265.22	209	65.54	30941	1965.45	159	86.8
stereovision2210	121576	5806.53	480	212.96	134006	6292.84	550	292.07
stereovision2308	238414	10487.42	812	377.91	229232	10208.72	770	548.46
Average	**1**	**1**	**1**	**1**	**0.92**	**0.96**	**0.91**	**1.46**

Table 6: Comparison between Min-cut + SA and Our Proposed approach

Casename	Min-cut + SA				Our Approach			
	WL	Delay	CrossD	Time	WL	Delay	CrossD	Time
bgm128	15599	1871.5	94	33.84	11962	1722.75	58	28.27
bgm224	58225	4360.36	291	75.12	41004	3752.51	217	63.64
bgm320	93559	6610.16	475	132.66	77292	6022.88	392	126.04
spree121	12794	960.43	295	64.8	9706	775.7	206	53.4
spree209	33301	2069.65	573	195.18	25840	1702.35	437	158.78
spree308	66616	3601.35	852	386.76	52813	3083.46	762	331.17
stereovision1105	23122	1234.99	216	78.96	16051	945.79	146	62.73
stereovision1210	90325	3970.49	566	236.1	56506	2772.1	416	204.56
stereovision1315	214682	8442.76	881	465.18	136079	5819.06	694	362.14
stereovision2112	38384	2221.66	201	67.2	33657	2017.12	143	58.59
stereovision2210	151469	6793.21	540	215.94	140514	6458.6	545	189.62
stereovision2308	215396	9542.86	581	345.54	247117	10784.21	839	313.57
Average	**1**	**1**	**1**	**1**	**0.79**	**0.86**	**0.83**	**0.85**

[5] T. Chen and Y. Chang. Modern Floorplanning Based on B*-Tree and Fast Simulated Annealing. *TCAD*, 2006.

[6] J. Cong and G. Luo. An analytical placer for mixed-size 3d placement. In *ISPD*, 2010.

[7] W. Donath. Complexity theory and design automation. In *DAC*, 1980.

[8] K. Eguro and S. Hauck. Enhancing timing-driven fpga placement for pipelined netlists. In *DAC*, 2008.

[9] A. Hahn Pereira and V. Betz. Cad and routing architecture for interposer-based multi-fpga systems. In *FPGA*, 2014.

[10] Y.-K. Ho and Y.-W. Chang. Multiple chip planning for chip-interposer codesign. In *DAC*, 2013.

[11] G. Karypis and etc. Multilevel hypergraph partitioning: Application in vlsi domain. In *DAC*, 1997.

[12] C. Kim and etc. Performance-driven circuit partitioning for prototyping by using multiple fpga chips. In *ASP-DAC*, 1995.

[13] S.-J. Lee and K. Raahemifar. Fpga placement optimization methodology survey. In *CCECE*, 2008.

[14] Q. Liu and etc. Ralp: Reconvergence-aware layer partitioning for 3d fpgas. In *ReConFig*, 2013.

[15] W.-H. Liu, M.-S. Chang, and T.-C. Wang. Floorplanning and signal assignment for silicon interposer-based 3d ics. In *DAC*, 2014.

[16] G. Marcel and A. Jason. Design Re-use for Compile Time Reduction in FPGA High-level Synthesis Flows. In *FPT*, 2014.

[17] F. Mo, A. Tabbara, and R. Brayton. A force-directed macro-cell placer. In *ICCAD*, 2000.

[18] H. Murata and etc. Rectangle-packing-based module placement. In *ICCAD*, 1995.

[19] J. Rose and et al. The VTR Project: Architecture and CAD for FPGAs from Verilog to Routing. In *FPGA*, 2012.

[20] K. Roy-Neogi and C. Sechen. Multiple fpga partitioning with performance optimization. In *FPGA*, 1995.

[21] D. Seemuth and etc. Automatic die placement and flexible i/o assignment in 2.5d ic design. In *ISQED*, 2015.

[22] M. Shih and etc. Performance-driven system partitioning on multi-chip modules. In *DAC*, 1992.

[23] K. Siozios and D. Soudris. A tabu-based partitioning and layer assignment algorithm for 3-d fpgas. *Embedded Systems Letters, IEEE*, 2011.

[24] T.-C. Wang and D. Wong. An Optimal Algorithm for Floorplan Area Optimization. In *DAC*, 1991.

[25] N.-S. Woo and J. Kim. An efficient method of partitioning circuits for multiple-fpga implementation. In *DAC*, 1993.

[26] Xilinx. Partial Reconfiguration of Xilinx FPGAs Using ISE Design Suite, 2012.

[27] Xilinx. White Paper: Vivado Design Suite, 2012.

[28] Xilinx. 7 Series FPGA Overview, 2013.

[29] M. Xu and etc. Near-linear wirelength estimation for fpga placement. In *CCECE*, 2009.

[30] P. Zhou and etc. 3d-staf: scalable temperature and leakage aware floorplanning for three-dimensional integrated circuits. In *ICCAD*, 2007.