TopoPart: a Multi-level Topology-Driven Partitioning Framework for Multi-FPGA Systems

Dan Zheng
The Chinese University of Hong Kong
dzheng@cse.cuhk.edu.hk

Xinshi Zang
The Chinese University of Hong Kong
xszang@cse.cuhk.edu.hk

Martin D.F. Wong
The Chinese University of Hong Kong
mdfwong@cuhk.edu.hk

2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD) | 978-1-6654-4507-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICCAD51958.2021.9643481

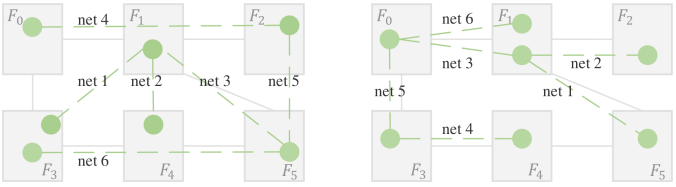
***Abstract*—As the complexity of circuit designs continues growing, multi-FPGA systems are becoming more and more popular for logic emulation and rapid prototyping. In a multi-FPGA system, different FPGAs are connected by limited physical wires, in other words, one FPGA usually has direct connections with only a few FPGAs. During the circuit partitioning stage, assigning two directly connected nodes to two FPGAs without physical links would significantly increase the delay and degrade the overall performance. However, some well-known partitioners, like hMETIS and PaToH, mainly focus on cut size minimization without considering such topology constraints of FPGAs, which limits their practical usage. In this paper, we propose a multi-level topology-driven partitioning framework, named as TopoPart, to deal with topology constraints in a multi-FPGA system. In particular, we firstly devise a candidate FPGA propagation algorithm in the coarsening phase to guarantee the later stages free of topology violations. In the last refinement phase, cut size is iteratively optimized maintaining both topology and resource constraints. Compared with the proposed baseline, our partitioning algorithm achieves zero topology violation while giving less cut size.**

***Index Terms*—multi-FPGA system, partitioning, topology**

I. Introduction

Nowadays, multi-FPGA systems have widespread applications in logic emulation and rapid prototyping of large designs because of their high flexibility and low cost [1]. A multi-FPGA system (MFS) consists of multiple FPGAs connected by physical wires or a programmable interconnection network [2]. In a typical compilation flow of an MFS, partitioning need be performed in the early stage to divide a massively large circuit into relatively smaller sub-circuits which are then assigned to different FPGAs. With circuit systems becoming increasingly huge and sophisticated, partitioning plays an ever more important role in system performance and delay.

In a typical MFS, the FPGAs can be heterogeneous and not identical because of the different available resources [3]–[5]. Besides, in most cases, FPGAs are not fully connected due to limited I/O resources and the connections between different FPGAs could be irregular. If an inter-FPGA signal starts from one source FPGA and heads for the other destination FPGA that is not physically connected with the source one directly, one or more intermediate FPGAs will be used as hops in the routing path [6]. These hops will bring much longer delay to the signal path and will also increase the number of inter-FPGA signals competing for the physical wires, which further deteriorates system delay. Therefore, during the step of partitioning, ignoring the topology of the MFS could have a significantly negative impact on system performance.



(a) Without topology (b) With topology

Fig. 1. The partitioning results without / with considering topology.

A toy example is given in Figure 1 to demonstrate the importance of this topology-aware partitioning. With the same MFS, two different partitioning results without and with considering the topology are shown in Figure 1(a) and 1(b) respectively. There are six inter-FPGA circuit nets denoted by dash lines. Although these nets are all cut, but the number of extra hops required differs a lot. In Figure 1(a), all nets except net 3 need routing through other FPGAs while all the nets are connected directly in Figure 1(b). Apparently, Figure 1(b) should be more preferred for better system performance.

In addition, some FPGAs in an MFS may have some specific functionality and different available resources [5]. It is thus common that some circuit modules need be assigned to some particular FPGAs in advance as fixed nodes for partitioning.

However, to the best of our knowledge, there is no existing research on multi-FPGA partitioning algorithms that consider the aforementioned characteristics of an MFS, i.e. topology driven partitioning with fixed nodes. The traditional and well-known partitioner hMETIS [7] mainly focuses on minimizing cut size with balancing constraints but not topology-driven nor addressing fixed node constraints. Another popular partitioner PaToH [8] allows fixed nodes but still cannot support topology constraints. The topology of an MFS is taken into consideration in [6], however, the main focus of their work is on routing across the FPGAs instead of partitioning. There are several recent works [1], [9]–[12] on multi-FPGA partitioning problem. The work [1] studies the partitioning problem by focusing on updating the net weights and generating results by hMETIS. The work [9] improves system performance by adding some constraints, such as combinatorial path and logical resource limitation, to the Wasga partitioning tool [10]. The work

978-1-6654-4507-8/21/$31.00 ©2021 IEEE

[11] explores two different partitioning methods, including a multilevel approach and a hierarchical approach, to minimize the number of cut nets in the partitioning result. The authors in [12] propose a simultaneous partitioning and signal grouping algorithm to achieve less signal delay. However, none of these recent works consider the connections between FPGAs or fixed node requirements during the stage of partitioning.

In this work, we propose a multi-level topology-driven partitioning framework, named TopoPart, to tackle the topology and fixed node constraints simultaneously. First, a candidate FPGA propagation algorithm is proposed, based on a guidance theorem, to efficiently calculate the potential FPGAs to place each circuit node. Then, candidate FPGAs will be considered and updated throughout the following multi-level stages, including coarsening, initial partitioning, uncoarsening and refinement. In particular, during coarsening, both connectivity between circuit nodes and the number of common candidate FPGAs are considered when merging two nodes. During initial partitioning, circuit nodes are assigned greedily to the candidate FPGA with minimum cut size and a traceback operation is designed to undo illegal assignments. In the final uncoarsening and refinement stage, cut size will be further minimized by iteratively moving circuit nodes to their candidate FPGAs under resource constraints. To evaluate the performance of our algorithm, we generate sufficient benchmarks based on several public circuits and FPGA benchmarks. We also devise a reasonable baseline for comparison by first applying PaToH to generate partitioning results with fixed nodes and then applying simulated annealing to reduce the topology violations. Compared with this baseline, our TopoPart can achieve zero topology violation with a smaller cut size. The main contributions of this paper are summarized as follows:

- We propose the first multi-level partitioning framework honoring both the topology and fixed node requirements of an MFS;
- We design novel and efficient algorithms to initialize and maintain candidate FPGA information in the multi-level partitioning framework, which can guarantee no topology violation;
- Experimental results show that our framework can achieve topology violation free results while satisfying fixed node constraint and the cut size is small.

The rest of this paper is organized as follows. Sec. II will introduce the terminology and problem. Sec. III will explain the details of the proposed algorithm. Sec. IV will discuss the benchmarks and experimental results. Finally, a conclusion is drawn in Sec. V.

II. Preliminary

In this section, we will formulate the topology-driven partitioning problem for multi-FPGA systems. Important terms and their meanings are summarized in Table I.

TABLE I
Terminology

Term	Description
$G(E,V)$	The circuit graph, E denotes circuit edge set, V denotes circuit node set
V_γ	The set of fixed circuit nodes
V_λ	The set of movable circuit nodes
$\hat{G}(\hat{E},\hat{V})$	The MFS graph, $\hat{E}$ denotes MFS edge set, $\hat{V}$ denotes FPGA node set
r_i	The resource capacity of FPGA $\hat{v}_i$
p_i	The set of circuit nodes assigned to FPGA $\hat{v}_i$
P	A partition solution $\{p_i \vert i = 1, 2, \ldots, \vert\hat{V}\vert\}$
$dist(v_i, v_j)$	Minimum distance between v_i and v_j on the graph G
$dist(\hat{v}_i, \hat{v}_j)$	Minimum distance between $\hat{v}_i$ and $\hat{v}_j$ on the graph $\hat{G}$
$part(v_i)$	The FPGA that the circuit node v_i is assigned to, $part(v_i) \in \hat{V}$
$Cddt(v_i)$	The set of candidate FPGA nodes that circuit node v_i can be assigned to
$Nbrs(v_i)$	The set of neighbour circuit nodes of v_i

A. Problem Formulation

The topology-driven partitioning problem can be formulated as follows:

Given a circuit graph $G(E,V)$, an FPGA graph $\hat{G}(\hat{E},\hat{V})$, and a set of fixed nodes V_γ, find a partitioning P such that the cut size is minimized while satisfying:

1) Each fixed node is assigned to its corresponding FPGA (fixed node constraint);
2) For each FPGA $\hat{v}_i$, the number of assigned nodes can not exceed the resource capacity r_i ($|p_i| \leq r_i$) (resource constraint);
3) For each two-pin net, either these two circuit nodes v_i and v_j are assigned to the same FPGA ($part(v_i) = part(v_j)$), or they are assigned to directly connected FPGAs ($(part(v_i), part(v_j)) \in \hat{E}$) (topology constraint).

III. Methodology

In this section, we will describe the proposed topology-driven partitioning framework, called TopoPart, in detail. Like hMETIS [7] and PaToH [8], TopoPart also adopts a multi-level framework to handle large-scale circuits but an important difference is that both topology and fixed node constraints are considered with high priority. The flow of TopoPart is shown in Figure 2. With the input circuit graph, FPGA graph, and the fixed node constraints, we first perform a candidate FPGA propagation algorithm to find the candidate FPGAs for each node while honoring both topology and fixed node constraints. Coarsening is then performed to merge different nodes that not only have high connectivity but also share a certain number of common candidate FPGAs into a single super-node. Next, an initial partitioning is conducted on the coarsest circuit graph to get a feasible initial solution without any violation. Finally, uncoarsening and move-based refinement are applied to optimize cut size.

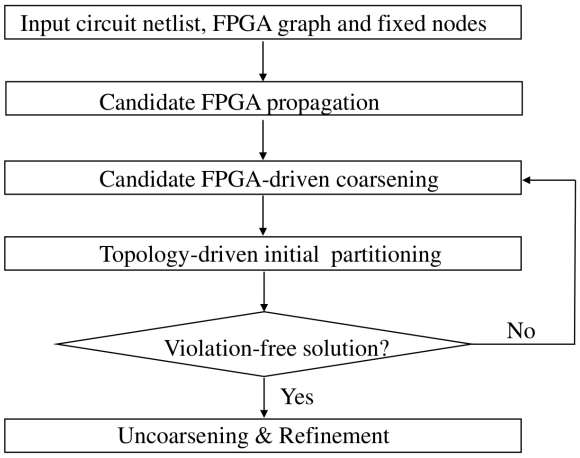


Fig. 2. Flow of the proposed topology-driven partitioning framework

A. Candidate FPGA Propagation

The existence of fixed nodes and the MFS topology limit the assignments of the movable circuit nodes. The set of FPGAs that a node v_i can be assigned to is called candidate FPGA set $Cddt(v_i)$ of v_i. Before the description of candidate FPGA generation algorithm, Theorem III.1 is introduced as follows.

Theorem III.1. *Let the length of the shortest path between v_i ($\hat{v}_i$) and v_j ($\hat{v}_j$) on the circuit graph G (FPGA graph $\hat{G}$) be x (y). The if-and-only-if condition that v_i and v_j can be assigned to $\hat{v}_{\hat{i}}$ and $\hat{v}_{\hat{j}}$ respectively without topology violation is $x \geq y$.*

Proof. First, we prove the only-if condition. Suppose v_i and v_j can be assigned to $\hat{v}_{\hat{i}}$ and $\hat{v}_{\hat{j}}$ respectively in a legal solution without topology violation. Assume $y > x$. Let p be a shortest path between v_i and v_j in the circuit graph. Let this path be $\{v_i, v_{\pi(1)}, \cdots, v_{\pi(x-1)}, v_j\}$ and let $\hat{v}_{\hat{\pi}(k)}$ be the FPGA to which node $v_{\pi(k)}$ is assigned in the legal solution. Since this is a legal solution, neighboring nodes on p are either on the same FPGA or assigned to neighboring FPGAs with a physical link in between. That means, there is a path connecting $\hat{v}_{\hat{i}}$ and $\hat{v}_{\hat{j}}$ in the FPGA graph of length at most x, which is contradictory to the assumption that $y > x$.
Next, we prove the if-direction. Suppose $x \geq y$. Let p and $\hat{p}$ denote the shortest paths between v_i and v_j and between $\hat{v}_{\hat{i}}$ and $\hat{v}_{\hat{j}}$ in the circuit graph and the FPGA graph respectively. There are $x+1$ circuit nodes on p and $y+1$ FPGA nodes on $\hat{p}$. A simple solution without topology violation can be constructed by successively assigning the circuits nodes on p to the FPGA nodes on $\hat{p}$, with v_i and v_j assigned to $\hat{v}_{\hat{i}}$ and $\hat{v}_{\hat{j}}$ respectively and the extra $x-y$ circuit nodes will all be assigned to $\hat{v}_{\hat{j}}$.

□

Algorithm 1 shows the procedure of candidate FPGA propagation. In the beginning, the maximum of the minimum distances with other nodes in the FPGA graph $maxDist(\hat{v}_i)$ is calculated for each FPGA node $\hat{v}_i$ using the Dijkstra's

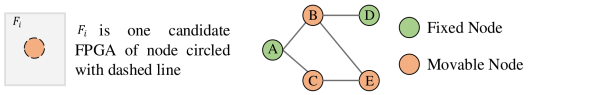
Algorithm 1: Candidate FPGA Propagation

Input: FPGA graph $\hat{G}(\hat{E}, \hat{V})$,
Circuit graph $G(E, V)$,
Fixed node-FPGA pair queue Q
Output: Candidate FPGA set $Cddt(v_i)$ for each node v_i

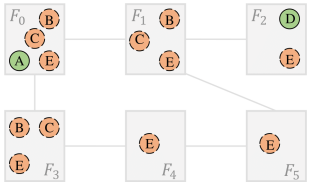
1 $maxDist(\hat{v}_i) \leftarrow \max_{j=1}^{|\hat{V}|} dist(\hat{v}_i, \hat{v}_j)$, $\forall \hat{v}_i \in \hat{V}$
2 **foreach** *FPGA node* $\hat{v}_i \in \hat{V}$ **do**
3 **for** $x = 1, 2, \ldots, maxDist(\hat{v}_i)$ **do**
4 $\hat{S}(\hat{v}_i, x) \leftarrow \{\hat{v}_j \in \hat{V} : dist(\hat{v}_i, \hat{v}_j) \leq x\}$
5 **foreach** *fixed node-FPGA pair* $(v_i, \hat{v}_{\hat{i}}) \in Q$ **do**
6 $d \leftarrow maxDist(\hat{v}_{\hat{i}})$
7 $S(v_i, d) \leftarrow \{v_j \in V : dist(v_i, v_j) < d\}$
8 $Cddt(v_i) \leftarrow \{\hat{v}_{\hat{i}}\}$, $\forall\ (v_i, \hat{v}_{\hat{i}}) \in Q$
9 $Cddt(v_i) \leftarrow \hat{V}$, $\forall v_i \in V_\lambda$
10 **while** $Q.size() > 0$ **do**
11 $(v_i, \hat{v}_{\hat{i}}) \leftarrow Q$.pop()
12 $d \leftarrow maxDist(\hat{v}_{\hat{i}})$
13 **foreach** *movable node* $v_j \in S(v_i, d)$ **do**
14 $k \leftarrow dist(v_j, v_i)$
15 $Cddt(v_j) \leftarrow Cddt(v_j) \cap \hat{S}(\hat{v}_{\hat{i}}, k)$
16 **if** $Cddt(v_j).size() == 1$ **then**
17 $\hat{v}_{\hat{j}} \leftarrow Cddt(v_j)$.top()
18 Add node pair $(v_j, \hat{v}_{\hat{j}})$ into Q
19 **if** $Cddt(v_j).size() == 0$ **then**
20 Return // no feasible solution

algorithm. Then, from line 2 to 7, for each FPGA node $\hat{v}_i$, the set $\hat{S}(\hat{v}_i, x)$ is constructed in such a way that it contains all the FPGA nodes whose distance with $\hat{v}_i$ is not larger than x. Similarly, for each fixed node-FPGA pair $(v_i, \hat{v}_{\hat{i}})$, with $d = maxDist(\hat{v}_{\hat{i}})$, the set $S(v_i, d)$ is composed of all the circuit nodes whose distance with v_i is smaller than d, in the circuit graph. Initially, for each fixed node, its candidate FPGA is just the pre-assigned FPGA and for each movable node, all FPGAs are its candidate FPGAs.

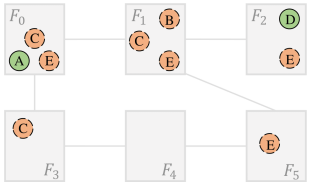
From line 10 to 20, the candidate FPGA sets of movable nodes are updated by combining the influence of all fixed nodes. Based on Theorem III.1, for each movable node v_j, the $Cddt(v_j)$ has nothing to do with a fixed node v_i *iff* $dist(v_j, v_i) \geq maxDist(\hat{v}_{\hat{i}})$, where $\hat{v}_{\hat{i}}$ is the FPGA node containing the fixed node v_i. In other words, only the movable node v_j satisfying $dist(v_j, v_i) < maxDist(\hat{v}_{\hat{i}}) = d$, i.e., $v_j \in S(v_i, d)$, needs to update its $Cddt(v_j)$ according to $\hat{v}_{\hat{i}}$. The updated candidate FPGA set $Cddt(v_j)$ will become the intersection between the original $Cddt(v_j)$ and $\hat{S}(\hat{v}_{\hat{i}}, k)$ with $k = dist(v_j, v_i)$. Note that $Cddt(v_j)$ could be influenced by more than one fixed node. If after multiple updates, $Cddt(v_j)$ only contains one candidate FPGA $\hat{v}_{\hat{j}}$, v_j will become a new fixed node on FPGA $\hat{v}_{\hat{j}}$ and the corresponding node pair $(v_j, \hat{v}_{\hat{j}})$ will be added into the queue Q for later propagation.



(a) A circuit graph.



(b) Candidate FPGAs for each node after propagation from fixed node A.



(c) Candidate FPGAs for each node after propagation from fixed node D.

Fig. 3. An example of candidate FPGAs propagation.

However, if $Cddt(v_j)$ is empty, there will be no feasible solution without topology violation.

A toy example is shown in Figure 3 to demonstrate the candidate FPGA propagation. Given an MFS graph and a circuit graph shown in Figure 3(a) and 3(b), fixed node A and D need to be placed on FPGA F_0 and F_2. The movable nodes B, C and E need to get their candidate FPGAs from propagation. As shown in Figure 3(b), starting from fixed node A, since $dist(A, B) = 1$, $Cddt(B)$ is $\hat{S}(F_0, 1) = \{F_0, F_1, F_3\}$. The same is true for C. For E, since $dist(A, E) = 2 \geq maxDist(F_0) = 2$, A does not affect $Cddt(E)$ and hence $Cddt(E)$ is still $\hat{V} = \{F_0, F_1, F_2, F_3, F_4, F_5\}$. We then consider the other fixed node D based on the current results. Since $dist(D, B) = 1$, $Cddt(B)$ will be $\{F_0, F_1, F_3\} \cap \hat{S}(F_2, 1) = \{F_1\}$. Because $dist(D, C) = 3 \geq maxDist(F_2) = 3$, D does not affect $Cddt(C)$ and $Cddt(C)$ is still $\{F_0, F_1, F_3\}$. Finally, since $dist(D, E) = 2$, $Cddt(E) = \hat{V} \cap \hat{S}(F_2, 2) = \{F_0, F_1, F_2, F_5\}$. Figure 3(c) displays the final candidate FPGAs for each node after propagation based on fixed node A and D.

With our propagation algorithm, both fixed nodes and topology constraints are well captured in candidate FPGA sets, which will provide global information for the later multi-level partitioning.

B. Candidate FPGA-driven Coarsening

In traditional multi-level partitioning methods, such as hMETIS [7] and PaToH [8], coarsening is the first stage where nodes with higher connectivity will be merged as a new super-node in the next level to reduce the solution space. Note that the typical connectivity between nodes is defined based on the net weight. The net weight is set to be one for each net by default.

However, this coarsening method cannot be applied directly to tackle the topology and fixed node constraint. The reason is that if two nodes have high connectivity but have very few or even no common candidate FPGAs, merging these two nodes will result in less feasible solutions or even infeasible solutions.

Therefore, the connectivity and the number of common candidate FPGAs should be considered simultaneously during the coarsening stage. In our algorithm, a parameter β is used to control the minimum number of common candidate FPGAs for coarsening. For two nodes, no matter how high their connectivity is, they will not be merged if the number of common candidate FPGAs is less than β. The selection of β is a trade-off between efficiency and quality. If β is too large, only a few nodes can be merged and coarsening will not work well, affecting inversely the efficiency of the method. On the other hand, if β is too small, the merged nodes may only have very few candidate FPGAs, which will very likely result in an infeasible solution after the coarsening stage. In our experiment, β is initially set as $\frac{2}{3} * |\hat{V}|$. Note that the candidate FPGAs of new super-nodes are the common part of those of merged nodes.

As mentioned, coarsening may remove some solutions, and it thus cannot be guaranteed to find a feasible solution even if one exists after coarsening. As shown in Figure 2, if no feasible solution can be found in the initial partitioning stage, the coarsening stage will be performed again with a larger parameter β.

C. Topology-driven initial partitioning

After coarsening is finished, the size of the circuit graph is significantly scaled down. Topology-driven initial partitioning is then applied in the final coarsened circuit graph to find a feasible solution with a small cut size.

Algorithm 2 describes this initial partitioning. The fixed nodes are first assigned to their corresponding FPGAs and then, from line 3 to 8, for each movable node v'_j, its partitioning status is labeled as -1 to indicate not being assigned to any FPGA yet and two relevant priority queues Q and R_j are maintained. Here, Q is used to decide the order to assign the movable nodes, in which the nodes with fewer candidate FPGAs will be assigned with higher priority. R_j is used to decide which FPGA the node v'_j should be assigned to first, where an FPGA will have a higher priority if assigning v'_j to it can bring a smaller cut size increment. Each time one movable node v'_j is selected and assigned to FPGA $\hat{v}_{\hat{j}}$. In each step of the loop from line 14 to 20, every neighboring node v'_k of v'_j that has not been assigned yet will update its candidate FPGA set and priority queue R_k by removing the FPGAs not connected to $\hat{v}_{\hat{j}}$ directly. In line 17, if there is no candidate FPGA for v'_k, which means v'_k cannot be assigned to any FPGA based on the current partitioning result, the traceback step will be activated to recover the status of v'_j and other relevant attributes. To improve the updating efficiency, we will not update the candidate FPGAs for other non-neighboring nodes because the update is not immediately needed and can be deferred until one of their own neighboring nodes is assigned to an FPGA.

In the process of tracing back, from line 21 to 27, v'_j is reset to be unassigned and the FPGA $\hat{v}_{\hat{j}}$ needs to be removed from the candidate FPGAs of v'_j. Additionally, the affected candidate FPGA sets of the neighbouring nodes of v'_j should be recovered. Here, we propose an efficient method to perform such recovery. For each unassigned node v_i, a vector T_i of

Algorithm 2: Topology-driven Initial Partitioning

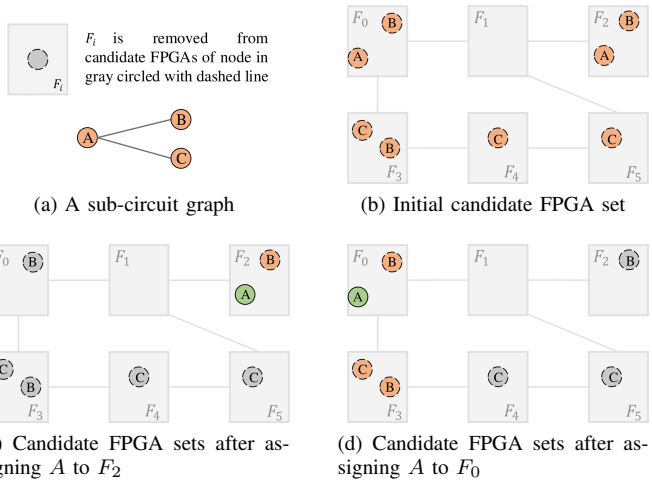
Input: Final coarsened circuit graph $G^{'}(E^{'}, V^{'})$,
Fixed node-FPGA pair set F,
Priority queue R_i in order of cut size for $\forall\ v_i' \in V^{'}$,
Priority queue Q in order of the number of candidate FPGAs

Output: Partition solution P

1 **foreach** *fixed node-FPGA pair* $(v_i', \hat{v}_{\hat{i}}) \in F$ **do**
2 $part(v_i') \leftarrow \hat{v}_{\hat{i}}$
3 **foreach** *movable node* $v_j' \in V^{'}$ **do**
4 $part(v_j') \leftarrow -1$
5 Add pair $(|Cddt(v_j')|, v_j')$ into Q
6 **foreach** *FPGA node* $\hat{v}_{\hat{j}} \in Cddt(v_j')$ **do**
7 Calculate cut size increment $\Delta(\hat{v}_{\hat{j}})$ if assigning v_j' to $\hat{v}_{\hat{j}}$
8 Add pair $(\Delta(\hat{v}_{\hat{j}}), \hat{v}_{\hat{j}})$ into R_j
9 **while** $Q.size() > 0$ **do**
10 $v_j' \leftarrow Q$.pop()
11 $\hat{v}_{\hat{j}} \leftarrow R_j$.pop()
12 $part(v_j') \leftarrow \hat{v}_{\hat{j}}$
13 $Traceback \leftarrow false$
14 **foreach** *unassigned node* $v_k' \in Nbrs(v_j')$ **do**
15 Update $Cddt(v_k')$ and R_k
16 **if** $|Cddt(v_k')| == 0$ **then**
17 $Traceback \leftarrow true$
18 Break `// Traceback is activated`
19 **else**
20 Add $(|Cddt(v_k')|, v_k')$ into Q
21 **if** $Traceback$ **then**
22 Remove $\hat{v}_{\hat{j}}$ from $Cddt(v_j')$
23 $part(v_j') \leftarrow -1$
24 Add $(|Cddt(v_j')|, v_j')$ into Q
25 **foreach** *unassigned node* $v_k' \in Nbrs(v_j')$ **do**
26 Recover $Cddt(v_k')$ and R_k
27 Add $(|Cddt(v_k')|, v_k')$ into Q

size $|\hat{V}|$ is used to record its candidate FPGAs. Initially, if v_i can be assigned to FPGA $\hat{v}_{\hat{l}}$, $T_i[\hat{l}]$ is 1; otherwise, $T_i[\hat{l}]$ is 0. During the update of $Cddt(v_k')$ on line 15, if an FPGA $\hat{v}_{\hat{i}}$ needs to be removed from $Cddt(v_k')$, the corresponding value of $T_k[\hat{i}]$ will be decreased by 1. Similarly, during the recovery on line 26, the corresponding value of $T_k[\hat{i}]$ will be increased by 1. By maintaining in such a way, the candidate FPGA set of the affected nodes can be updated and recovered quickly in $O(m)$ time, where m is the number of FPGAs in an MFS.

This initial partitioning with traceback is illustrated by an example shown in Figure 4. In Figure 4(a), we select a sub-circuit netlist containing three nodes for a simple illustration. As shown in Figure 4(b), the initial candidate FPGA sets



(a) A sub-circuit graph

(b) Initial candidate FPGA set

(c) Candidate FPGA sets after assigning A to F_2

(d) Candidate FPGA sets after assigning A to F_0

Fig. 4. An example of initial partitioning with traceback

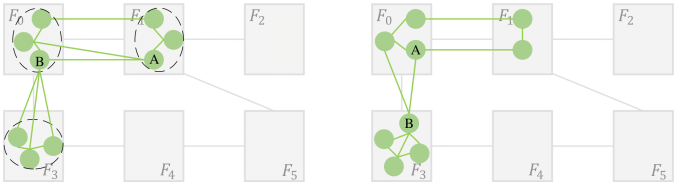
for A, B, and C are $\{F_0, F_2\}$, $\{F_0, F_2, F_3\}$ and $\{F_3, F_4, F_5\}$ respectively, which are represented by vectors $[1,0,1,0,0,0]$, $[1,0,1,1,0,0]$, and $[0,0,0,1,1,1]$. Node A will be assigned first since it has the least number of candidate FPGAs. Suppose assigning A to FPGA F_2 produces less cut size, A is temporarily assigned to F_2 as in Figure 4(c). F_0, F_3, F_4 and F_5 should then be removed from the candidate FPGA sets of B and C. The updated candidate FPGA sets of B and C will then be $[0,0,1,0,-1,-1]$ and $[-1,0,0,0,0,0]$ by subtracting $[1,0,0,1,1,1]$ from them. As there is no candidate FPGA for C, the current partitioning will violate topology constraints. Therefore, traceback needs to be performed by unassigning A from FPGA F_2, and the candidate FPGA sets of B and C should be recovered to the original ones by adding $[1,0,0,1,1,1]$ back to them. In Figure 4(d), without the choice of F_2, A can only be assigned to F_0 and the candidate FPGA sets of B and C are updated as $\{F_0, F_3\}$ and $\{F_3\}$ respectively. Apparently, C can only be placed at F_3 and B will prefer F_0 due to a smaller cut size.

D. Uncoarsening and Refinement

After a feasible solution of the final coarsened circuit graph is obtained by initial partitioning, uncoarsening and move-based refinement will be performed level by level. The objective of refinement is to minimize the cut size while maintaining the MFS topology and resource constraints.

At each level, the nodes will be uncoarsened from the super-nodes in the previous level and the partitioning results in the previous level will also be inherited as the initial solution. The boundary nodes, which are nodes connected with a node on another FPGA, will be marked as the candidate nodes to move. For each boundary node v_i, all the potential FPGAs that v_i can be moved to without violating any constraints will be checked. If the cut size can be reduced, v_i will be moved to the FPGA such that the cut size is minimized the most.

Figure 5 is an example of uncoarsening and refinement. In this example, nine nodes are uncoarsened from three super-



(a) Before refinement (b) After refinement

Fig. 5. An example of uncoarsening and refinement

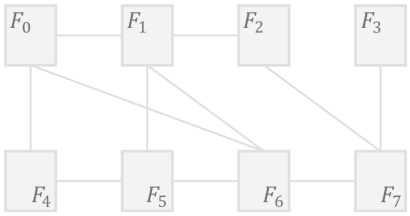


Fig. 6. First multi-FPGA system MFS1

nodes in the previous level and node A, B are two boundary nodes that can reduce cut size. Node B cannot be moved at the beginning due to the topology constraints. Hence, A will firstly be moved to FPGA F_0 with cut size decreased by one and then B can be moved to FPGA F_3 to further improve the cut size.

IV. Experiment

In this work, all algorithms are implemented in C++ and tested on a Linux workstation with an Intel Xeon 3.4GHz CPU with 8 cores and 32GB memory. Note that, all the experiments are conducted under the limitation of 8 threads. Since no previous work considers the topology of the FPGA graph in the circuit partitioning problem, there are no benchmarks nor baseline that can be directly used for experimental comparisons. For empirical study, we generated two sets of benchmarks[1] based on a public FPGA benchmark suite and some synthetic circuits which will be discussed in Sec. IV-A. We have also designed a baseline by combining PaToH and simulated annealing for a fair comparison which will be discussed in Sec. IV-B. Detailed experimental results and analysis will be discussed in Sec. IV-C.

A. Benchmarks

Two multi-FPGA systems named MFS1 and MFS2 are used for evaluation, which both come from Problem B of the 2019 CAD Contest at ICCAD [13]. MFS1 is a simple MFS with 8 FPGAs and 11 connections as shown in Figure 6. MFS2 is more complex with 43 FPGAs and 214 connections among these FPGAs. With sparsity defined as connection number divided by all-pair connection number, the sparsity of MFS2 is 0.24 and that of MFS1 is 0.39, which means MFS2 is much sparser than MFS1.

As for the circuit, we first adopt the Titan23 benchmark suite [14] whose statistics are summarized in Table II. The "# Nodes" represents the number of circuit nodes, the "# Nets" denotes the number of nets in the circuit netlist, and the "# Fixed" means the total number of fixed nodes. Note that, we convert the original n-pin nets to $(n-1)$ two-pin nets by the star model, i.e., connecting the source pin to the destination pins in order to handle the topology constraints. Thus, the "# Nets" is the number of two-pin nets after this modelling. Besides, because of lacking fixed nodes in the original Titan23 benchmark, we created these fixed nodes by (1) finding an initial feasible solution by iterative node movement and (2) randomly picking several nodes on each FPGA as fixed nodes. In MFS1, we assign three fixed nodes on each FPGA and there are 24 fixed nodes in total. Unlike MFS1, we cannot find any feasible initial solution without topology violation using the Titan23 benchmark for the FPGA system MFS2. Therefore, to make sure the existence of a feasible solution to test with the MFS2 system, we generated another 8 synthetic benchmarks in the following way. First, nodes are randomly and evenly assigned to the FPGAs. Two-pin nets are then generated according to a net-to-node ratio of 2.5 in our experiment. About one-fifth of the nets are inter-FPGA nets or cut nets connecting nodes on FPGAs having direct physical links. Finally, on each FPGA, one to three nodes are selected randomly to be fixed. The statistics of these benchmarks are shown in Table III.

[1]We make these generated benchmarks public at https://drive.google.com/drive/folders/14cXR0dZA-3H5BY0BcZ6KFvDf4E1NeTon?usp=sharing.

B. Baseline

As there is no existing research on circuit partitioning with FPGAs' topology constraints, to make a fair comparison, we devised a reasonable baseline based on PaToH [8] and simulated annealing. Since PaToH could honor fixed node requirements, it will be invoked at the beginning to generate an initial partitioning solution without considering the topology constraints. Simulated annealing will then be applied in a post-processing stage to iteratively reduce the topology violations. In addition, the initial feasible solutions used to generate benchmarks are regarded as our reference solution.

C. Experimental Results

In Table II and III, the "Cut Size" denotes the number of two-pin nets that are cut and the "# Violations" represents the number of cut nets violating topology constraints. The "Avg. Ratio" of "# Violations" is the average ratio of violated cut nets to total cut nets for all the benchmarks. The "Avg. Ratio" of "Cut size" and "Time" is the average relative performance of TopoPart comparing with "PaToH + Simulated Annealing" for all the benchmarks.

As shown in Table II, in the first experiment with Titan23 benchmarks on MFS1, the reference solutions are feasible solutions without any topology violations but with enormous and worst cut size in all the benchmarks. Compared with the reference solutions, PaToH with simulated annealing can dramatically reduce the cut size by 6.34 times in ratio. However, in this proposed baseline, minimizing cut size at the beginning

TABLE II
STATISTIC OF TITAN23 BENCHMARKS AND RESULTS OF DIFFERENT METHODS ON MFS1.

Benchmark	# Nodes	# Nets	# Fixed	PaToH + Simulated Annealing			Reference		TopoPart		
				Cut Size	# Violations	Time (s)	Cut Size	# Violations	Cut Size	# Violations	Time (s)
sparcT1_core	91976	411095	24	74981	24014	19.305	271165	0	54841	0	86.342
neuron	92290	327876	24	17017	336	16.412	207742	0	8815	0	23.692
stereo_vision	94059	287174	24	66871	10404	21.748	178068	0	32068	0	55.616
des90	111221	631452	24	112982	14115	13.936	457301	0	87211	0	126.112
SLAM_spheric	113115	495472	24	45233	8339	5.567	330033	0	54402	0	90.432
cholesky_mc	113250	477730	24	36859	4694	30.773	293245	0	37658	0	51.957
segmentation	138295	622376	24	31973	393	4.042	403018	0	34092	0	48.945
bitonic_mesh	192064	1129440	24	190693	35837	22.318	834725	0	184894	0	149.847
dart	202354	909795	24	206519	42086	36.476	613436	0	168197	0	251.163
openCV	217443	989365	24	34299	9038	54.61	719543	0	120428	0	117.595
stap_qrd	240240	970335	24	364997	8972	33.182	572275	0	134256	0	296.304
minres	261359	988482	24	196331	24065	63.633	707162	0	96178	0	145.168
cholesky_bdti	266422	1067463	24	421706	6842	49.072	641868	0	75479	0	110.059
denoise	275638	1253869	24	44496	6013	9.997	815159	0	93464	0	115.618
sparcT2_core	300109	1313186	24	116730	17990	50.756	884700	0	98128	0	185.788
gsm_switch	493260	2204961	24	442141	45736	152.823	1399933	0	267424	0	471.599
mes_noc	547544	2609889	24	519114	94634	105.536	1783947	0	297140	0	372.403
LU_Network	574369	2327102	24	218046	39712	195.488	1745314	0	152508	0	407.184
LU230	635456	2604888	24	450199	47464	317.847	1679962	0	134877	0	261.887
sparcT1_chip2	820886	3210647	24	503334	120764	269.555	2075194	0	207814	0	517.124
directrf	931275	3437764	24	914431	28986	137.222	2235984	0	113493	0	248.817
bitcoin_miner	1089284	3109046	24	650207	23131	907.746	2070060	0	413566	0	1231.155
Avg. Ratio				1	0.13	1	6.34	0	0.84	0	4.85

TABLE III
STATISTIC OF 8 SYNTHETIC BENCHMARKS AND RESULTS OF DIFFERENT METHODS ON MFS2.

Benchmark	# Nodes	# Nets	# Fixed	PaToH + Simulated Annealing			Reference		TopoPart		
				Cut Size	# Violations	Time (s)	Cut Size	# Violations	Cut Size	# Violations	Time (s)
case1	300000	749091	84	133613	85697	21.40	149800	0	149065	0	208.55
case2	400000	998902	83	178223	130765	34.35	199876	0	162611	0	254.55
case3	500000	1247856	81	211268	157081	45.13	249952	0	139118	0	300.45
case4	600000	1498396	87	264195	193117	59.36	299814	0	350546	0	419.71
case5	700000	1748250	81	313037	226607	71.34	349890	0	177704	0	375.81
case6	800000	1998061	78	367156	280786	89.34	399966	0	149259	0	402.34
case7	900000	2247701	90	384340	276000	104.77	449828	0	304398	0	551.28
case8	1000000	2497555	79	476251	295670	119.60	499904	0	270777	0	550.91
Avg. Ratio				1	0.71	1	1.12	0	0.79	0	6.32

does not help to reduce the topology violations in the later post-processing stage. As a result, there are still a large number of topology violations, with the violated cut nets accounting for about 13% of the total cut nets. What is worse, reducing topology violations in the post-processing stage could affect inversely the cut size minimized by PaToH. Compared with the proposed baseline, our TopoPart can consider minimizing cut size and satisfying topology constraints at the same time. Therefore, TopoPart not only can keep topology violation free but also reduce further 16% cut size. TopoPart has longer running time, but all the results satisfy the topology constraints and it is still reasonably fast for all test cases.

The results of the second experiment with synthetic datasets on MFS2 are shown in the Table III. Since MFS2 is much larger and sparser than MFS1, the topology constraints of the FPGAs could have more impacts on different methods. Unlike in MFS1, the partitioning results generated by the reference solution in MFS2 can have better cut size that is designed to account for about 20% of total nets. Compared with the reference, the proposed baseline can bring small improvement on the cut size but still has a large number of topology violations. The violated cut nets comprise almost 71% of the total cut nets, which means most of the nets connecting different FPGAs violate the topology constraints. However, TopoPart can again out-perform the proposed baseline with less cut size and zero topology violations, which shows the superiority of our method in the larger and sparser MFS.

V. CONCLUSION

This work proposes TopoPart, a multi-level topology-driven partitioning framework for multi-FPGA systems, to improve the system performance by taking the connections between FPGAs into consideration. To guarantee no topology violation, the candidate FPGA set for each movable circuit node is initialized with a candidate FPGA generation algorithm and maintained in all stages of the multi-level framework. Through empirical study, our TopoPart has shown outstanding performance in tackling the topology constraint while minimizing cut size compared to the proposed baseline.

VI. Acknowledgement

The authors sincerely thank Prof. Evangeline F.Y. Young for her careful guidance and valuable suggestions.

References

[1] S.-H. Liou, S. Liu, R. Sun, and H.-M. Chen, "Timing driven partition for multi-fpga systems with tdm awareness," in *Proceedings of the 2020 International Symposium on Physical Design*, 2020, pp. 111–118.

[2] W.-S. Kuo, S.-H. Zhang, W.-K. Mak, R. Sun, and Y. K. Leow, "Pin assignment optimization for multi-2.5 d fpga-based systems," in *Proceedings of the 2018 International Symposium on Physical Design*, 2018, pp. 106–113.

[3] S. Hauck, G. Borriello, and C. Ebeling, "Mesh routing topologies for multi-fpga systems," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 6, no. 3, pp. 400–408, 1998.

[4] U. Farooq, H. Mehrez, and M. K. Bhatti, "Comparison of direct and switch-based inter-fpga routing interconnect for multi-fpga systems," in *2017 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. IEEE, 2017, pp. 1–6.

[5] M. Gort and J. H. Anderson, "Analytical placement for heterogeneous fpgas," in *22nd international conference on field programmable logic and applications (FPL)*. IEEE, 2012, pp. 143–150.

[6] U. Farooq, H. Mehrez, and M. K. Bhatti, "Inter-fpga interconnect topologies exploration for multi-fpga systems," *Design Automation for Embedded Systems*, vol. 22, no. 1, pp. 117–140, 2018.

[7] G. Karypis, "hmetis 1.5: A hypergraph partitioning package," *http://www. cs. umn. edu/˜ metis*, 1998.

[8] Ü. V. Çatalyürek and C. Aykanat, "Patoh (partitioning tool for hypergraphs)," in *Encyclopedia of Parallel Computing*. Springer, 2011, pp. 1479–1487.

[9] M. Turki, H. Mehrez, Z. Marrakchi, and M. Abid, "Partitioning constraints and signal routing approach for multi-fpga prototyping platform," in *2013 International symposium on system on chip (SoC)*. IEEE, 2013, pp. 1–4.

[10] "Wasga partitioning tool," https://www.flexras.com/.

[11] U. Farooq and B. A. Alzahrani, "Exploring and optimizing partitioning of large designs for multi-fpga based prototyping platforms," *Computing*, vol. 102, no. 11, pp. 2361–2383, 2020.

[12] S.-C. Chen, R. Sun, and Y.-W. Chang, "Simultaneous partitioning and signals grouping for time-division multiplexing in 2.5 d fpga-based systems," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–7.

[13] Y.-H. Su, R. Sun, and P.-H. Ho, "2019 cad contest: System-level fpga routing with timing division multiplexing technique," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–2.

[14] K. E. Murray, S. Whitty, S. Liu, J. Luu, and V. Betz, "Timing-driven titan: Enabling large benchmarks and exploring the gap between academic and commercial cad," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 8, no. 2, pp. 1–18, 2015.