An Efficient Hypergraph Partitioner under Inter-Block Interconnection Constraints

1st Benzheng Li
School of Microelectronics
Xidian University
Xi'an, China
0000-0002-1227-3909

2nd Hailong You
School of Microelectronics
Xidian University
Xi'an, China
0000-0003-3427-5320

3rd Shunyang Bi
School of Microelectronics
Xidian University
Xi'an, China
0009-0001-1341-3977

4th Yuming Zhang
School of Microelectronics
Xidian University
Xi'an, China
0000-0002-8587-0747

***Abstract*—Multi-FPGA systems are increasingly employed for very large scale integration circuit emulation and prototyping. Due to limited I/O resources, each FPGA often only has direct physical connections to a few other FPGAs. Therefore, if signals between FPGAs originate from a source FPGA and flow toward a target FPGA not directly connected to the source FPGA, intermediate FPGAs will be used as hops in the signal path. These FPGA-hops increase signal delays and the number of physical lines used in signal multiplexing between FPGAs, degrading system performance. To address these issues, researchers proposed partitioners that guarantees zero hop, but they lead to a considerable cut-size. In this paper, building on previous research, we introduce a new candidate block propagation theorem and optimize the initial partition process based on its corollary. Additionally, we also present a method for correcting violations during uncoarsening to improve the solver capability. Results of experiments demonstrate that our proposed method significantly reduces the cut size by 96% while retaining comparable running times.**

***Index Terms*—partition, multi-FPGA system, hardware emulation**

I. Introduction

Field programmable gate arrays (FPGAs), with their programmable nature, are commonly employed as fundamental platforms in prototyping and hardware emulation. Nowadays, as the explosion of circuit design complexity and the limited capacity of individual FPGAs, multi-FPGA systems (MFSs) have emerged as the primary solution for addressing the challenges imposed by large-scale designs. The partitioning stage is a critical step in the MFS compilation process, as it ensures that each sub-design is assigned to the appropriate FPGA.

The circuit net-lists can be easily represented as hypergraphs and partitioned using hypergraph partitioning algorithms [1]. As an well-known NP-hard problem [2], hypergraph partitioning has always been a hot research topic. The Kernighan Lin (KL) algorithm [3] and Fiducia Mattheyses (FM) algorithm [4] were the initial graph partitioning algorithms based on iterative node swapping, and their variants have been widely adopted in various partitioners. The hMETIS [5] along with subsequent algorithms like PaToH [6], UMPa [7] and KaHyPar [8] employ a multi-level framework, significantly improving the speed and quality of partitioning. After incorporating network flow and undergoing multiple improvement [9], KaHyPar has emerged as the state-of-the-art hypergraph partitioner.

This workwas supported by National Natural Science Foundation of China (Grant No. 62234010), the 111 Project of China (Grant No. 61574109), and the Science and Technology Development Program of Shaanxi (Grant No.2023-YBGY-273).

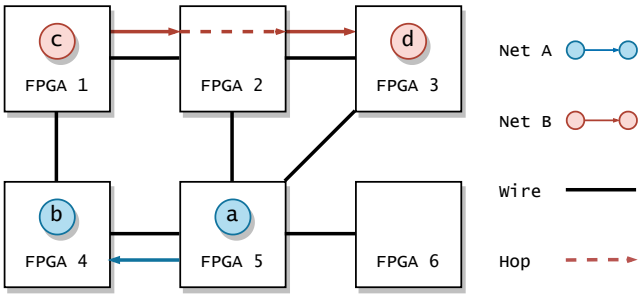


Fig. 1: A multi-FPGA system with 6 FPGAs and 7 physical wires.

These traditional hypergraph partitioning algorithms primarily focus on the total cut-size. However, MFS introduces additional constraints due to the special topological structure, which presenting greater challenges in solving the hypergraph partitioning problem. Due to limited I/O resources, in most cases, each FPGA is only directly connected to a few other FPGAs. If the signal between FPGAs starts from a source FPGA and flows to a target FPGA that is not directly connected to the source FPGA, one or more intermediate FPGAs will be used as a relay in the signal path. As shown in Figure 1, since there is no wire between FPGA $(1, 3)$, the signal in net B needs to go through one FPGA-hop to complete communication. This hop behavior will make the signal path longer and increase the number of signal multiplexing, thereby increasing the system latency by several times. In conclusion, it is crucial to deeply consider the impact of the FPGA interconnection topology in MFS to truly enhance the operating frequency of the system.

In recent years, several research works have focused on the interconnection relationship of FPGAs. Liou et al. [10] and SPARK [11] tried to refine the time-division multiplexing (TDM) ratio and maximum distance of hops after partitioning. Their method has reduced the system delay compared to traditional hypergraph partitioner. By observing their partitioning results, it can be concluded that hop distance is the main factor that worsens delay. Therefore, some scholars focus on more radical attempts to solve this problem. They try to complete the partitioning without generating any hops. Li et al. [12]

979-8-3503-4859-0/DATE24/© 2024 EDAA

introduced a partitioner with interconnection constraint that considers candidate FPGA of node during coarsening and initial partitioning phase. TopoPart [13] proposed a propagation algorithm to efficiently maintain candidate FPGA of each node. However, the solving ability of these algorithms is still not strong enough, and in challenging large-scale circuit designs, it is easy to result in the inability to find a solution with zero hops.

In response to the above issues, this paper presents an efficient hypergraph partitioner for resolving the inter-block interconnection constraints. The objective is to minimize the cut-size while avoiding communication between FPGAs that are not directly connected. This algorithm takes into account the FPGA connection topology at the partitioning stage and accomplishs the partitioning and system-level placement simultaneously. It includes a coarsening and initial partitioning based on candidate blocks, as well as a local improvement stage with violation corrections. The main innovations of our work are summarized as follows:

1) We introduce an important propagation theorem and its corresponding corollary, which significantly accelerates the candidate block propagation process. The usage of this corollary enables real-time maintenance of candidate block without significantly increasing time complexity, thus improving the ability to find feasible solutions.
2) We propose a post-processing violation correction method to further improve solving ability.
3) Experimental results show that our partitioner reduced by several orders of magnitude in terms of cut-size when compared with previous work.

The remainder of this paper is organized as follows. Section II defines the partitioning problem under inter-block interconnection constraints and presents an abstract representation of the constraints and objectives. In Section III, we present the high-performance zero-hop partitioner. Section IV provides details on the experimental setup and presents the results, demonstrating the effectiveness and efficiency of our proposed algorithms. Finally, Section V concludes the paper.

II. Problem Formulation

Given a net-list represented by hypergraph $H(V, E)$ and a multi-FPGA system model represented by graph $G(\hat{V}, \hat{E})$. The gates or units of circuit are represented by node $v_i \in V$ and nets are represented by hyperedges $e_i \in E$. Each FPGA is depicted by the block $\hat{v}_i \in \hat{V}$, and edge $\hat{e}(\hat{u}, \hat{v}) \in \hat{E}$ represents the physical wire between FPGA $\hat{u}$ and $\hat{v}$. The demanded resource on FPGA for gate i is represented as n_i, while the resource capacity of FPGA i is denoted as r_i. The number of signals that need to be communicated in each net is represented as the weight ω_{ei} of hyperedge e_i.

In a circuit netlist, each net consists of a driving gate and several driven gates. The logic levels are emitted by the driving gate and transmitted to the driven gates. Similarly, each hyperedge contains a source node and several drain nodes. The signal of the hyperedge starts from the source node and propagates to each drain node. The source node of e_i is denoted as $s(e_i)$, while the drain nodes are represented by $d(e_i)$. The $b(u)$ indicates the block where node u is placed. $\hat{U}(e) = \{b(a_i)|a_i \in d(e)\}$ represents the set of partitions where the drain nodes in super edge e belong.

TABLE I: Important notation used in this work.

Variable	Definition
$H(V, E)$	Circuit netlist represented by hypergraph
v_i	Gate represented by node i
e_i	Net represented by hyperedge i
$b(v_i)$	FPGA where gate i is located
$s(e_i)$	Source node of net i
$d(e_i)$	Set of Drain nodes of net i
$G(\hat{V}, \hat{E})$	MFS represented by block graph
$\hat{v}_i$	FPGA represented by block i
$\hat{e}_i(\hat{u}, \hat{v})$	Physical wire between $\hat{u}$ and $\hat{v}$ represented by edge i
$\hat{U}(e)$	The set of FPGAs where the drain gates in net e belong
$dis(u, v)$	Shortest path length in H between u and v
$\widehat{dis}(\hat{u}, \hat{v})$	Shortest path length in G between $\hat{u}$ and $\hat{u}$
n_i	Needed resource of FPGA i
r_i	Supplied resource of FPGA i
$CB(u)$	Set of candidate blocks of node u

Suppose all hyperedges in E have a length of 1. In this case, the shortest path length between u and v is defined as $dis(u, v)$. Similarly, if all edges in $\hat{E}$ have a length of 1, the shortest path length between $\hat{u}$ and $\hat{v}$ is defined as $\widehat{dis}(\hat{u}, \hat{v})$.

The following describes how the objective function is calculated. Let $cutsize(e)$ denote the cut cost of hyperedge e, then it can be expressed as:

$$cutsize(e) = \omega_e \times \left|\hat{U}(e)\right| \tag{1}$$

The problem of hypergraph partitioning under inter-block interconnection constraints is formulated as follows: Given the H and G, the algorithm needs to determine the block where each node is located. The objective function can be formulated as follows:

$$\begin{aligned} \text{minimize} \quad & F = \sum_{e \in E} cutsize(e) \\ \text{subject to } & n_i \le r_i, i = 1, \ldots, |\hat{V}| \\ & \forall e \in E,\ \hat{e}(b(s(e)), b(d_i(e))) \in \hat{E} \end{aligned} \tag{2}$$

Table I provides a list of the important notations used throughout this work.

III. Algorithm

TopoPart [13] is a partitioner that ensures the partition schemes generated do not include any cut edges that require block-hop. While it fulfills our functional requirements, it does not fully meet our standards of speed and partition quality. To address the shortcomings, we developed an upgraded partitioner that also guarantees zero hop count. Its performance is significantly improved by several orders of magnitude compared to TopoPart.

A. Candidate Block Propagation

In order to achieve zero-hop partitioning, Zheng et al. [13] proposed a partitioner called TopoPart, which employs a concept known as "Candidate FPGA Propagation". This idea begins to take effect when a part of the nodes has been placed. To avoid hop, the remaining nodes can only be assigned to certain specific partitions called "candidate blocks" (CBs). The set of blocks that a node x can be assigned to is represented as $CB(x)$. To filter out partitions where nodes cannot be placed legally, the authors proposed a theorem. Under the problem definition in this paper, the theorem can be described as follows:

Theorem 1. *For nodes $a, b \in V$ and blocks $\hat{u}, \hat{v} \in \hat{V}$, the if-and-only-if condition that a and b can be assigned to $\hat{u}$ and $\hat{v}$ respectively without any hop occurring is $dis(a,b) \geq \widehat{dis}(\hat{u},\hat{v})$.*

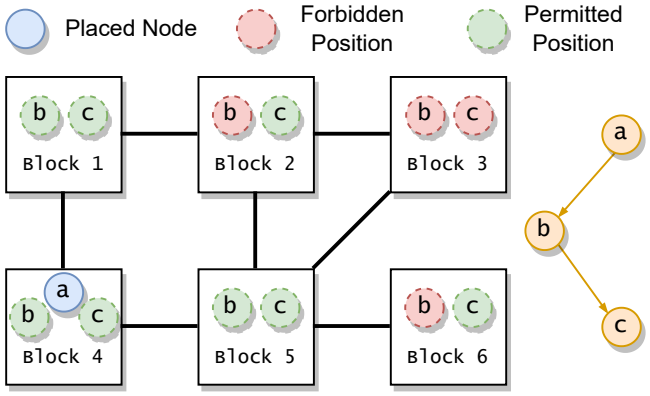


Fig. 2: Candidate blocks schematic diagram.

For instance, as illustrated in Figure 2, nodes a, b, and c are interconnected as the right-hand side hypergraph shows. Suppose node a has been placed in Block 4. In accordance with Theorem 1, nodes b and c cannot be assigned to the block denoted by the gray node, otherwise there will inevitably be cut edges between blocks without wires. This behavior, which is not permitted in this article, is called "cut violation" (CV). The algorithm maintains the CB of every node effectively during the coarsening and initial partitioning phases. During coarsening, the algorithm ensures that the number of CBs for each cluster exceeds a designated threshold. Additionally, the initial partitioning algorithm prioritizes nodes with fewer CBs. By leveraging this approach, the problem-solving capability of the partitioner is significantly enhanced.

However, according to the study in [13], maintaining the CB set is a time-consuming task. When placing a node x, the process of PROPAGATE(x) is executed to select the candidate block of the node. This process is called "propagation", and its pseudocode is shown in Algorithm 1.

Line 5 defines the set $S(a,d)$, which saves all nodes whose shortest path length to node a is no more than d. It was obtained through a breadth-first search (BFS) starting from a. In the case of a relatively dense hypergraph H, the number of nodes in $S(a,d)$ may be very large. Therefore, executing the statements in lines 6-13 for all these nodes will consume a lot of time.

Algorithm 1 PROPAGATE(x)

Require: $H(V,E)$, $G(\hat{V},\hat{E})$, the node to be propagated is x, the block of x is $\hat{x}$.

1: $d \leftarrow \max\{\widehat{dis}(\hat{x},\hat{v}) - 1 | \hat{v} \in \hat{V}\}$
2: $Q \leftarrow \{x\}$
3: **while** $Q \neq \emptyset$ **do**
4: $\quad a \leftarrow Q.pop$
5: $\quad S(a,d) \leftarrow \{b | dis(a,b) \leq d, b \in V\}$
6: $\quad$ **for all** movable node $b \in S(a,d)$ **do**
7: $\quad\quad k \leftarrow dis(a,b)$
8: $\quad\quad \hat{S}(\hat{u},k) \leftarrow \{\hat{v} | \widehat{dis}(\hat{u},\hat{v}) \leq k, \hat{v} \in \hat{V}\}$
9: $\quad\quad CB(b) \leftarrow CB(b) \cap \hat{S}(\hat{u},k)$
10: $\quad\quad$ **if** $|CB(b)| = 1$ **then**
11: $\quad\quad\quad Q \leftarrow Q \cup \{b\}$
12: $\quad\quad$ **if** $|CB(b)| = 0$ **then**
13: $\quad\quad\quad$ **return** *no feasible solution*

To overcome this problem, TopoPart adopts a compromise strategy. Specifically, before the clustering stage, it uses the complete propagation algorithm specifically for the initially fixed nodes. At the initial partitioning stage, the algorithm only considers the impact of placing nodes on their neighboring nodes, which is equivalent to fixing the value of d in line 1 of the pseudocode to 1. This means that some illegal blocks in $CB(a)$ have not been eliminated, which may lead to unsolvable situations in subsequent circumstances, triggering backtracking.

B. Fast Maintenance Algorithm for Candidate Blocks

This section introduces a pivotal propagation theorem and the corresponding corollary. These enable the algorithm to speed up the CB propagation process significantly, thereby facilitating real-time maintenance of CB without drastically increasing the time complexity. As a result, the ability to identify feasible solutions is enhanced considerably. The proposed theorem is as follows.

Theorem 2. *Let $a, b, c \in V$ and $\hat{u}, \hat{v}, \hat{w} \in \hat{V}$, a and b are placed on $\hat{u}$ and $\hat{v}$ respectively. If*

$$dis(a,b) + dis(b,c) < \widehat{dis}(\hat{u},\hat{w}) \tag{3}$$

then

$$dis(b,c) < \widehat{dis}(\hat{v},\hat{w}) \tag{4}$$

Proof. *Because $\widehat{dis}(\hat{u},\hat{w})$ represents the shortest path length between u and w, it does not exceed the length of path traversed by $\hat{v}$.*

$$\widehat{dis}(\hat{u},\hat{w}) \leq \widehat{dis}(\hat{u},\hat{v}) + \widehat{dis}(\hat{v},\hat{w}) \tag{5}$$

By combining this equation with Equation (3)*, we can obtain*

$$dis(a,b) + dis(b,c) < \widehat{dis}(\hat{u},\hat{v}) + \widehat{dis}(\hat{v},\hat{w}) \tag{6}$$

According to Theorem 1, the already placed node a and b satisfy $dis(a,b) \geq \widehat{dis}(\hat{u},\hat{v})$. On the left and right side of Equation (6)*, delete $dis(a,b)$ and $\widehat{dis}(\hat{u},\hat{v})$ respectively. The inequality still holds, and this results in Equation* (4)*.*

Corollary 1. *When generating set $S(a,d)$ using BFS, nodes that have already been propagated are not added to the BFS queue. The set obtained in this way is referred to as $S'(a,d)$. Updating CB using S' or S leads to completely identical results.*

Proof. *Assuming that node a is currently being propagated, node b has already been propagated, and node c is any other node. If the algorithm is running correctly, block $\hat{w}$ should be removed from $CB(c)$. Now, let's prove that even after ignoring node b in the BFS queue, block $\hat{w}$ still won't be present in $CB(c)$. We need to consider two cases here.*

1) *If node b is not on the shortest path from a to c, removing node b won't have any effect on node c, as a can reach c through other paths.*
2) *If node b is on the shortest path from a to c, then it must satisfy*

$$dis(a,b) + dis(b,c) = dis(a,c) \tag{7}$$

Since block $\hat{w}$ needs to be removed, we have

$$dis(a,c) = dis(a,b) + dis(b,c) < \widehat{dis}(\hat{u},\hat{w}) \tag{8}$$

According to Theorem 2, we must have

$$dis(b,c) < \widehat{dis}(\hat{v},\hat{w}) \tag{9}$$

This means that when PROPAGATE(b) *was executed in a previous run of the algorithm, block $\hat{w}$ had already been removed from $CB(c)$, so whether or not it is considered for removal during the current propagation process won't affect the result.*

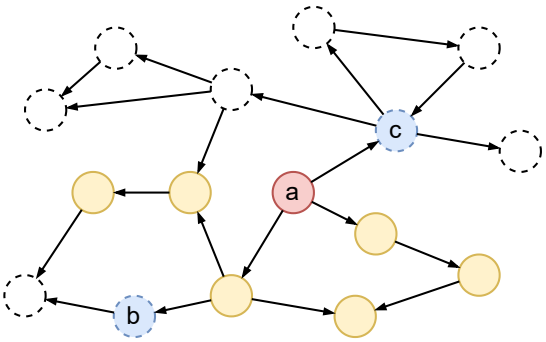


Fig. 3: The adoption of Corollary 1 reduces the count of nodes in $S(a,d)$.

Using Corollary 1 can significantly reduce the number of nodes that need to be enumerated. For instance, in Figure 3, it can be observed that when $d = 3$, all sixteen nodes are situated at a distance less than or equal to d from node a. By applying Corollary 1, assuming nodes b and c were propagated beforehand, they can be disregarded during the BFS process. Therefore, only six yellow nodes are necessitated in $S(a,d)$. When performing CB propagation on the TopoPart test case, the average number of nodes in $S(a,d)$ decreased by hundreds of times.

By applying Corollary 1 to the PROPAGATE(x) process, every node placed in the initial partitioning stage will undergo a complete propagation process. This method can eliminate more infeasible decisions while ensuring that the time complexity of the algorithm does not increase. This reduces the number of backtracking caused by unsolvable problems in subsequent processes, and in some cases, actually reduces the overall runtime.

C. Initial Partitioning Algorithm Based on Candidate Blocks

Now, the number of CBs is no longer the decisive criterion for determining the processing node sequence, but the weighted combination of the number of CBs and cut-size is considered. In the past, this method had a high risk of not finding a solution. However, thanks to the new real-time propagation strategy, the ability to find feasible solutions has been enhanced, allowing us to focus more on minimizing the cut-size.

The initial partitioning algorithm commonly uses the *Greedy Hypergraph Growing* algorithm. In traditional partitioners, this algorithm uses a priority queue to maintain the connectivity between each node-block pair. A typical way of computing the connectivity is as follows:

$$C(a,\hat{u}) = \sum_{\substack{e \ni a \\ e \cap N(\hat{u}) \neq \emptyset}} \omega_e \tag{10}$$

This means the sum of connection weights between this node and all nodes already assigned to the block. The algorithm selects the node-block pair with the highest connectivity and places the corresponding node in the corresponding block. This process continues until all nodes have been assigned to a block.

The proposed approach differs from the traditional approach in the prioritization of nodes based on their CBs. Firstly, each node can only be placed within its respective CB, or else it will result in a CV. Secondly, during placement, the connectivity calculation takes into account both the number of CBs and the connection weight. This is because nodes with fewer CBs have a smaller solution space and are more susceptible to becoming unsolvable in later stages. Therefore, such nodes are given higher priority during the placement process. The new connectivity calculation function is defined as follows:

$$C'(a,\hat{u}) = C(a,\hat{u}) + \frac{\alpha}{|CB(a)|} \tag{11}$$

Here, $|CB(a)|$ represents the number of CBs for node a and α is the weight factor for CB terms, which is empirically set to $100 \times |\hat{V}|$.

D. Local Refinement Algorithm with Cut Violation Correction

If placing node a in a partition causes the CB count of another node b to be 0, this placement will not be allowed. In the TopoPart algorithm, if a node cannot be accommodated within any CBs, the algorithm will terminate and report that there is no solution. This paper proposes a better strategy to find more hidden partition solutions in such cases.

In this section, we introduce a set R, which serves as a temporary holding area for nodes that should not be placed in any partition. The placing of nodes in R will cause the CB count of other nodes to be 0. The remaining nodes are processed in the order. Once all nodes have been processed, R contains

several nodes that still need to be allocated to blocks. The next step is to determine the optimal way to accommodate these unallocated nodes within the existing partitions.

The method is based on the *Greedy Hypergraph Growing* algorithm mentioned in Section III-C. This section proposes a new connectivity function and uses it to determine the placement of nodes in R. Define $vio(e)$ as the number of blocks in $\hat{U}(e)$ that are not directly connected to $b(s(e))$, which is called the CV count of e. Let $vio'(e, a, \hat{u})$ denote the CV count of e after placing node a in $\hat{u}$. In addition, introduce a penalty term $\Delta vio(a, \hat{u})$ to measure the change in CV generated after placing node a in $\hat{u}$. It is calculated as follows:

$$\Delta vio(a, \hat{u}) = \sum_{e \ni a} (vio'(e, a, \hat{u}) - vio(e)) \times \omega_e \tag{12}$$

To incorporate the importance of the CV factor in influencing the placement of nodes in R, we need to modify the connectivity function. The new connectivity function, denoted by $C^R(a, \hat{u})$, is calculated as follows:

$$C^R(a, \hat{u}) = C(a, \hat{u}) - \beta \Delta vio(a, \hat{u}) \tag{13}$$

where β is set to a very large value. This function determines the order of node-block pairs in the priority queue and reruns the initial partitioning to complete the placement of nodes in R. By following this process, the placement method that minimizes CV is obtained.

In the subsequent phase of local refinement, the algorithm centers its attention on correcting all CVs. This procedure relies on the FM refinement algorithm. Different from conventional algorithms, the gain function for node movement now incorporates a penalty term for CV modification, as presented in equation Equation (13). The importance of reducing the number of CVs supersedes all else at present, so this term has been given significant weight. If after uncoarsening and refinement, the number of CVs decreases to zero, this zero-hop partition is considered successful.

This method has been proven to be capable of rediscovering the solution in several challenging partition cases, thus avoiding the direct declaration of failure in finding the zero-hop partition plan.

IV. Experimental Results

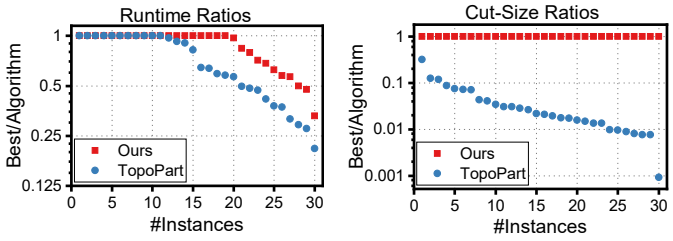
In this section, we will evaluate the performance of our partitioner. We implemented our work in C++ and compiled it using g++ 11.3.0. All experiments were conducted on a single core of a Linux workstation. The machine utilized an Intel Xeon Gold 6248R 3.0GHz CPU, running on Ubuntu 22.04 OS.

A. Benchmarks

The experiments were performed using the Titan23 benchmark suite [14] along with eight net-lists generated by [13]. The statistical information for these experiments can be found in Table II. To ensure consistency of experiment in TopoPart [13], we conducted the experiments using the following settings:

TABLE II: Result comparison on benchmarks.

Testcase	#Nodes	#Edges	TopoPart		Ours	
			Cut-Size	Runtime(s)	Cut-Size	Runtime(s)
sparcT1_core	91976	411084	54841	86.34	1185	35.93
neuron	92290	327876	8815	23.69	1110	19.52
stereo_vision	94050	287174	32068	55.61	480	16.18
des90	111221	631452	87211	126.11	2482	39.88
SLAM_spheric	113115	495472	54402	90.43	3915	25.02
cholesky_mc	113250	477730	37658	51.95	997	25.04
segmentation	138295	622376	34092	48.94	2542	28.34
bitonic_mesh	192064	1129440	184894	149.84	3193	88.74
dart	202354	909795	168197	251.16	53553	232.5
openCV	217453	989365	120428	117.59	980	74.71
stap_qrd	240240	970335	134256	296.30	1286	62.22
minres	261359	988471	96178	145.16	859	131.4
cholesky_bdti	266422	1067463	75479	110.05	1023	51.68
denoise	275638	1253869	93464	115.61	8200	57.48
sparcT2_core	300109	1313186	98128	185.78	958	105.02
gsm_switch	493260	2200864	267424	471.59	19054	178.27
mes_noc	547544	2609818	297140	372.40	4646	1131.24
LU_Network	635456	2604885	152508	407.18	17961	151.23
LU230	574372	2327101	134877	261.88	1024	419.55
sparcT1_chip2	820886	3144781	207814	517.12	2779	757.45
directrf	931275	3437764	113493	248.81	873	159.94
bitcoin_miner	1089284	3109045	413566	1231.15	378	1192.96
case1	300000	749091	149065	208.55	6469	215.28
case2	400000	998902	162611	254.55	4959	303.52
case3	500000	1248756	139118	300.45	5612	379.97
case4	600000	1498396	350546	419.71	6224	590.34
case5	700000	1748250	177704	375.81	5480	663.52
case6	800000	1998061	149259	402.34	5075	806.84
case7	900000	2247701	304398	551.28	6355	955.81
case8	1000000	2497555	270777	550.91	5233	1156.15
Ave. Ratio			1	1	0.044	0.97



(a) (b)

Fig. 4: Performance plots for TopoPart and our partitioner.

- Using the "SampleInput" from the ICCAD'19 Contest [15] as the FPGA interconnection constraint for the net-list of Titan23, which includes 8 FPGAs and 11 physical connections between them. We also use "synopsys01" as FPGA interconnection constraint for the net-lists generated by [13], which consists of 43 FPGAs and 214 connections.
- The original n-pin nets were transformed into $(n-1)$ two-pin nets using the star model.
- The same fixed node was set for each FPGA.
- The balance factor was ignored.

B. Results

Table II shows the cut-size and runtime of the partitions produced by TopoPart and our algorithm across all test cases. Both algorithms successfully obtained zero-hop partitioning solutions for all 30 cases. The experimental results of TopoPart were obtained directly from the research paper, but the nearly identical CPU performance ensured fairness in the comparison.

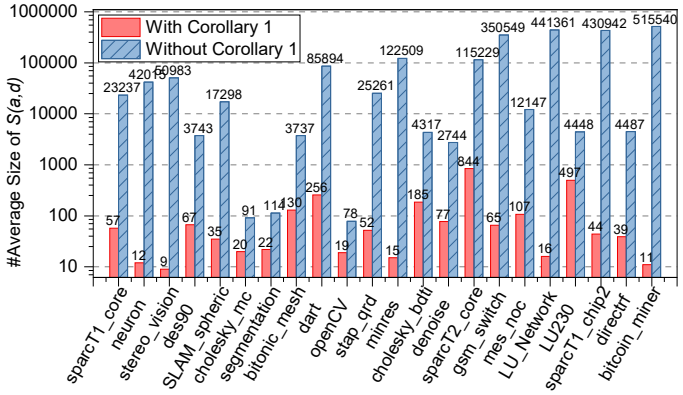


Fig. 5: The average size of $S(a, d)$ with or without Corollary 1 on Titan23 benchmark.

Figure 4 presents a more intuitive comparison of the results of the two methods in terms of cut-size and runtime. The x-axis corresponds to 30 test cases that were used, while the y-axis represents the ratio between the best score produced by both algorithms and the value generated by the respective algorithm (best value/algorithm value). The ratio of each algorithm is sorted in descending order. A ratio of one indicates that the algorithm produced the optimal solution. Lower curve points indicate poorer performance of the corresponding algorithm. Our algorithm demonstrated significant improvements in cut-size, with an average reduction to $0.04\times$ compared to TopoPart. These results indicate that performing propagation after each node placement can greatly enhance performance and optimize the partitioning process. Additionally, the approach of combining cut-size and CB count in the initial partition also helps to improve the quality of the solution. Additionally, the runtime of the partitioning has not only remained unchanged but slightly decreased.This is because complete propagation removes more partitions from the candidate block list of each node that will cause violations later, thus reducing the backtracking process caused by infeasibility. In additional, our algorithm consumes only 1.8 GB of peak memory on the largest case.

We applied Corollary 1 and performed a comparison between the time complexity with and without its application during the candidate block propagation process in the initial partition. To gain statistical insights, we calculated the average elements count of $S(a, d)$ in Algorithm 1. The results are visualized in the histogram depicted in Figure 5. Upon applying Corollary 1, the average size of $S(a, d)$ across all testcases decreased from 102,578 to 117. This reduction signifies a significant complexity reduction of $876\times$.

When "synopsys01" is used as the interconnect constraint of Titan23 net-list, the huge block network greatly increases the difficulty of solving problems. In the initial partitioning phase, no zero-violation solutions can be found for all cases. However, after the local refinement algorithm with CV correction that proposed in Section III-D, the number of violations dropped significantly, and "SLAM_spheric" and "segmentation" successfully eliminated all violations.

V. Conclusion

This paper proposes a high-performance circuit partitioning method that can avoid signal hops between FPGAs in MFS application scenarios. A new propagation theorem and its corollary are presented and proven, and experiments demonstrate that this discovery can reduce the time complexity of critical algorithms by orders of magnitude. The proposed method is applied in the initial partitioning stage after optimization, ensuring that the partitioner can rapidly acquire solutions with low cut cost. The paper also includes a post-processing method for violation repair, further enhancing the solving ability. In comparison with previous work, this method lowers cut-size by 96%.

Acknowledgment

The authors acknowledge Zheng et al. [13] for opening their testcases for comparison and clearing our doubts.

References

[1] R. Burra and D. Bhatia, "Timing driven multi-fpga board partitioning," in *Proceedings Eleventh International Conference on VLSI Design*. IEEE, 1998, pp. 234–237.

[2] P. Esrom, "Combinatorial algorithms for integrated circuit layout by t. lengauer john wiley and sons, chichester (uk), 1990, xxviii+ 286 pp.(£ 35)," *Robotica*, vol. 9, no. 1, pp. 118–118, 1991.

[3] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *The Bell system technical journal*, vol. 49, no. 2, pp. 291–307, 1970.

[4] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in *Papers on Twenty-five years of electronic design automation*, 1988, pp. 241–247.

[5] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: Application in vlsi domain," in *Proceedings of the 34th annual Design Automation Conference*, 1997, pp. 526–529.

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

[7] U. V. Catalyiirek, "Umpa: A multi-objective, multi-level partitioner for communication minimization umit v. catalyiirek, mehmet deveci, kamer kaya, and bora ucar," *Graph partitioning and graph clustering*, vol. 588, p. 53, 2013.

[8] Y. Akhremtsev, T. Heuer, P. Sanders, and S. Schlag, "Engineering a direct k-way hypergraph partitioning algorithm," in *2017 Proceedings of the Ninteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*. SIAM, 2017, pp. 28–42.

[9] S. Schlag, T. Heuer, L. Gottesbüren, Y. Akhremtsev, C. Schulz, and P. Sanders, "High-quality hypergraph partitioning," *ACM Journal of Experimental Algorithmics*, vol. 27, pp. 1–39, 2023.

[10] 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.

[11] X. Zang, E. F. Young, and M. D. Wong, "Spark: A scalable partitioning and routing framework for multi-fpga systems," in *Proceedings of the Great Lakes Symposium on VLSI 2023*, 2023, pp. 593–598.

[12] B. Li, Z. Qi, Z. Tang, X. He, and H. You, "High quality hypergraph partitioning for logic emulation," *Integration*, vol. 83, pp. 67–76, 2022.

[13] D. Zheng, X. Zang, and M. D. Wong, "Topopart: a multi-level topology-driven partitioning framework for multi-fpga systems," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–8.

[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.

[15] 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.