MFSPart: A Generalized Partitioning Framework for Multi-FPGA Systems and Its Ensemble-Based Extension

Yugao Zhu, Wenji Fang, Yao Lu, Shang Liu, Yanzhen Zhu, Qijun Zhang, and Zhiyao Xie, *Member, IEEE*

***Abstract*—Multi-FPGA systems (MFSs) are essential for prototyping very large-scale integration (VLSI) designs, necessitating efficient partitioning techniques. Optimizing inter-FPGA communication—particularly minimizing hop count—is critical for improving performance and reducing latency. However, existing topology-aware partitioning methods primarily minimize driver–sink pair cut size under a strict one-hop constraint, which may not always yield feasible solutions. They also typically overlook practical constraints such as FPGA capacity limits and I/O resource optimization, leading to solutions with excessive pin usage and routing complexity.**

To address these limitations, we propose a unified formulation that jointly optimizes driver–sink pair cut size, connectivity, and mean hop count under maximum-hop and capacity constraints. Unlike prior frameworks that enforce hop-based constraints via propagation before coarsening—imposing significant limitations—we postpone propagation until after coarsening. Based on this, we develop MFSPart, a framework applicable to both fixed-node and non-fixed-node scenarios, featuring improved coarsening, initial partitioning, and refinement strategies. We further propose an enhanced version, MFSPart-Ensemble, which systematically combines high-quality structures from diverse solutions across uncoarsening stages to generate superior partitions. Experiments demonstrate that MFSPart achieves excellent performance across all three objectives with reduced runtime, while MFSPart-Ensemble further improves solution quality, especially for complex FPGA systems.

***Index Terms*—Multi-FPGA system, hypergraph partitioning, hardware emulation**

I. Introduction

A. Background of Multi-FPGA System (MFS) Partitioning

With the development of very large-scale integration (VLSI) circuits, modern integrated circuits (ICs) typically comprise millions of logic elements. To handle this, multi-FPGA systems are gaining prominence in IC emulation and prototyping [1], [2] by partitioning large netlists across multiple FPGAs [3]–[6], necessitating customized partitioning algorithms.

Partitioning for MFSs differs significantly from traditional approaches (e.g., hMETIS [7], KaHyPar [8], and SpecPart [9]). MFS partitioning is a complex optimization problem that involves simultaneously considering multiple critical factors, such as system topology [10]–[12], FPGA capacity [13], [14], and I/O resource limitations [2], [15], facilitating the routing process [16]–[20] to optimize the signal path of every net. A key challenge in MFS partitioning lies in satisfying the low-hop requirement, which means that driver-sink pairs within a net should ideally be placed on **the same or nearby** FPGAs. Satisfying the low-hop requirement significantly reduces inter-FPGA communication delay, conserves I/O resources, and enhances overall system performance, a principle that has gained wide recognition in both industry and academia [12], [13], [21]–[26]. Following this principle, the methods in [12], [25] employ a "one-hop constraint", which restricts every driver-sink pair to **the same or adjacent** FPGAs. This constraint, as illustrated in Fig. 1, restricts the solution space and increases the difficulty of finding satisfiable solutions. Based on this constraint, these partitioning algorithms aim to minimize the *driver-sink pair cut size*—defined as the total number of driver-sink pairs spanning different FPGAs—thereby effectively reducing overall inter-FPGA latency.

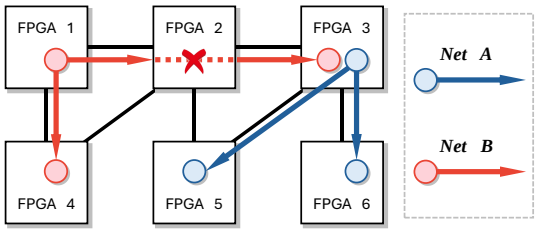


Fig. 1: A Multi-FPGA system example with 6 FPGAs and 7 interconnections. Net A satisfies the "one-hop constraint," whereas Net B violates the "one-hop constraint".

Existing research on MFS partitioning can be classified into two categories: the **fixed-node case** [12], [25], [26], where certain nodes are pre-assigned, and the **non-fixed-node case** [21], which extends the prior work [25] to scenarios without pre-assignments. Despite their different starting assumptions, these algorithms typically follow a general four-step framework: (1) *Propagation*: The framework first identifies initial fixed nodes and then determines a set of candidate FPGAs[1] for each unfixed node, as illustrated in Fig. 2. To enable propagation in the non-fixed-node case, the method in [21] selects one node for each FPGA to serve as an initial fixed node. These initial assignments then help to infer candidate FPGAs for the remaining nodes; (2) *Coarsening*: Adjacent nodes with many shared candidate FPGAs are merged, reflecting a high likelihood of being assigned to the same FPGA; (3) *Initial Partitioning*: Each unfixed node is sequentially assigned to one of its candidate FPGAs to generate an initial solution; and (4) *Uncoarsening & Refinement*: The partitioning solution is gradually projected onto finer hypergraphs, and refined by moving nodes from one FPGA to another at each level (e.g.,

This work is supported by Hong Kong Research Grants Council (RGC) YCRG Grant C6003-24Y, GRF Grant 16200724, and ACCESS – AI Chip Center for Emerging Smart Systems, supported by the InnoHK initiative of Innovation and Technology Commission of the Hong Kong Special Administrative Region Government.

Yugao Zhu, Wenji Fang, Yao Lu, Shang Liu, Yanzhen Zhu, Qijun Zhang, and Zhiyao Xie are with the Department of Electronic and Computer Engineering, Hong Kong University of Science and Technology, Hong Kong (Corresponding author: Zhiyao Xie).

[1]The FPGAs that do not cause the node to violate a given topology constraint (e.g., the "one-hop constraint").

TABLE I: Comparison of the problem formulation between existing works and our methods.

Method	Constraints			Objective		
	Allow Over One Hop	Explicit FPGA Capacity	Fixed Node Scenario	Driver-sink Pair Cut Size	Connectivity Metric	Mean Hop Count
[12]	✗	✗	Fixed	✓	✗	✗
[25]	✗	✗	Fixed	✓	✗	✗
[21]	✓	✓	Fixed&Non-Fixed	✓	✗	✓*
[26]	✓	✗	Fixed	✓	✓*	✗
MFSPart MFSPart-Ensemble	✓	✓	**Fixed& Non-Fixed**	✓	✓	✓

* While these metrics are considered in prior work, [21] optimizes the *mean hop count* only after its partitioning is complete. Meanwhile, [26] only uses the *driver-sink pair cut size* in a scenario comparable to ours and applies the *connectivity metric* in other contexts.

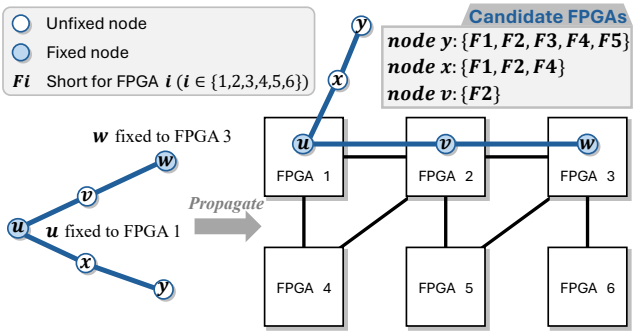


Fig. 2: The propagation for satisfying the "one-hop constraint" in existing works: calculating candidate FPGAs (CF) for each unfixed node $\{v, x, y\}$, based on fixed nodes $\{u, w\}$.

with the Fiduccia-Mattheyses (FM) algorithm [27]) to further improve solution quality.

Our in-depth study of the works [1], [3], [12], [21]–[26], [28]–[31] in MFS partitioning reveals limitations in both the problem formulation and the methodologies. Therefore, we aim to propose a more applicable problem formulation in Sec. I-B. Correspondingly, in Sec. I-C, we briefly introduce a new partitioning framework, **MFSPart**, along with its extension, **MFSPart-Ensemble**[2], to guide partitioning in the multi-FPGA system domain.

B. Problem Formulation Improvement

We propose a more comprehensive and flexible low-hop-oriented formulation that improves signal delay and I/O resource efficiency, which will be formally introduced in Sec. II.

Our approach incorporates several realistic constraints including (1) FPGA capacity limits, (2) a user-defined maximum hop-count threshold, and (3) optional fixed-node assignments. At the same time, we jointly optimize three key metrics: (1) the *driver-sink pair cut size*, (2) the *connectivity metric*, and (3) the *mean hop count*. Table I provides a comparison of our constraints and optimization objectives with related works. Each constraint and objective is discussed below.

Constraint 1: Low-hop constraint. A key limitation of works enforcing a strict "one-hop constraint" [12], [25] is that this constraint is not always satisfiable, as evidenced by the experimental results in [12]. To address this rigidity, we introduce a more flexible and practical "low-hop constraint". This constraint stipulates that the hop count of any driver-sink pair must not exceed a user-defined maximum threshold $H_{\max}$, where $H_{\max} \geq 1$, relaxing the strict requirement from prior works.

Constraint 2: FPGA capacity constraint. Unlike existing methods [12], [25] that assume infinite FPGA capacity[3], our framework explicitly enforces realistic FPGA capacity limits, ensuring the feasibility of the resulting partitions.

Constraint 3: Fixed-node constraint (optional). Practical designs often require certain components, such as debugging interfaces, to be assigned to specific FPGAs [5]. While existing works are often tailored to a single scenario, our formulation accommodates both configurations.

Objective 1: Driver-sink pair cut size. The latency of inter-FPGA signal transmission in a netlist is typically much higher than that of intra-FPGA signals [24]; consistent with prior works [12], [21], [25], [26], we adopt the *driver-sink pair cut size* as a primary objective to reduce signal delay.

Objective 2: Connectivity metric. To optimize I/O resource utilization, we minimize the *connectivity metric* [26], [32], defined as the number of FPGAs connected by a net minus one. Minimizing this metric encourages assigning more sink (receiver) nodes of a net onto the same FPGA, thereby reducing I/O resource usage and simplifying routing.

Objective 3: Mean hop count. To more comprehensively capture system latency under our flexible "low-hop constraint", we introduce the *mean hop count* as an objective. This metric complements the *driver-sink pair cut size* by directly optimizing the average signal path length across the multi-FPGA system.

C. Our New Frameworks: MFSPart and MFSPart-Ensemble

Beyond introducing a generalized problem formulation, our framework also presents significant algorithmic advancements. As Fig. 3 summarizes, we find 4 limitations of the existing MFS partitioning paradigm in [12], [21], [25], [26] and make 4 corresponding key enhancements. The first 3 enhancements comprise our standard MFSPart solution, simultaneously achieving better *driver-sink pair cut size*, *connectivity metric*, and *mean hop count* with shorter runtime than the state of the art. The 4$^{\text{th}}$ enhancement leads to a more advanced version named MFSPart-Ensemble, generating even better partitioning solutions by devoting a longer time.

Limitation 1: Evolving candidate information & costly propagation. In existing works, propagation is performed

[2]Our source code is available at https://github.com/hkust-zhiyao/MFSPart.

[3]We confirmed this by inspecting prior solutions in detail and communicating with some of the authors. Despite assuming infinite capacity, placing all nodes in one FPGA is infeasible due to initial fixed nodes in specific FPGAs.

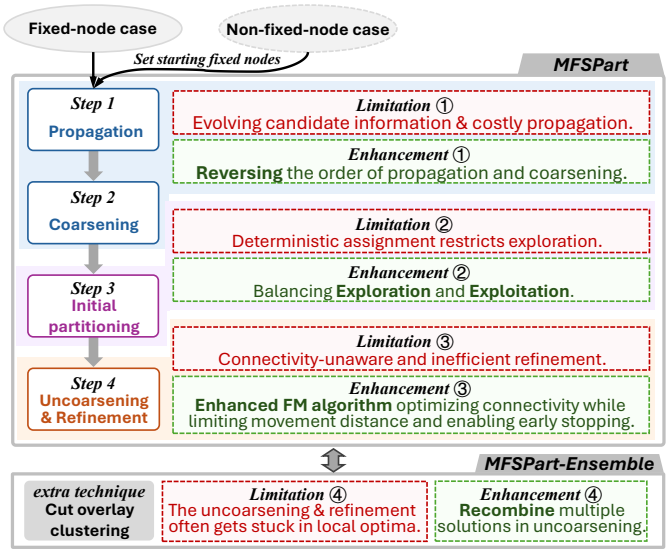


Fig. 3: The overview of the existing MFS partitioning framework, its limitations and our corresponding enhancements.

prior to coarsening, as the coarsening process relies on each node's candidate FPGA information, which must be derived via propagation. However, the candidate FPGA information may evolve as coarsening proceeds because coarsening alters the hypergraph topology, potentially leading to violations in the subsequent partitioning phase[4]. Moreover, performing propagation on the original hypergraph is time-consuming.

Enhancement 1. We address Limitation 1 by reversing the order of propagation and coarsening. Specifically, we execute coarsening directly, followed by propagation on the coarsest hypergraph. Deferring propagation to the coarsest hypergraph not only ensures that the candidate FPGA information remains accurate for the subsequent initial partitioning phase but also significantly reduces runtime. This novel order eliminates the dependency on candidate FPGA information during coarsening, which we achieve via our **m-coarsen** and **a-coarsen** methods that rely on nodal affinity to preserve topology.

Limitation 2: Deterministic assignment restricts exploration. The initial partitioning of existing frameworks typically generates a deterministic solution using a greedy approach. As a result, it often tends to assign most nodes to a single FPGA to minimize cut size, thereby limiting the exploration of the broader solution space.

Enhancement 2. To address Limitation 2, our hybrid initial partitioning strategy effectively balances exploration and exploitation by integrating probabilistic and deterministic methods. This approach increases the flexibility of partitioning and promotes diverse, high-quality initial solutions, comprehensively considering factors such as capacity, cut size, violations, and the degree of each FPGA (i.e., its number of FPGA interconnections).

Limitation 3: Connectivity-unaware and inefficient refinement. The refinement phase in existing MFS partitioning suffers from critical shortcomings. 1) These refinement approaches are generally connectivity-unaware, leading to increased I/O resource usage and routing complexity. 2) The FM operations in [25], [28] treat all violating node pairs[5] equally, failing to penalize longer violations more heavily[6]. 3) The refinement phase dominates the overall runtime, as it evaluates the gain of moving nodes to all potential FPGAs. Therefore, accelerating this phase is essential.

Enhancement 3. We address these limitations by proposing an enhanced FM algorithm with three advancements. 1) We redesign the gain function to incorporate the *connectivity metric*, directly optimizing for I/O usage. 2) We introduce a hop-based refinement metric that assigns larger penalties to more distant violating pairs, thereby handling violations more effectively. 3) We improve runtime efficiency by limiting the node movement distance and applying an "early stopping" criterion.

Limitation 4: The uncoarsening & refinement often gets stuck in local optima. The final limitation lies in the multi-level framework itself, which serves as the backbone of both existing works and our MFSPart. Specifically, the uncoarsening & refinement phase is inherently prone to getting trapped in local optima, often resulting in only incremental improvements over iterations. The "low-hop constraint" in MFS partitioning further exacerbates this challenge. Consequently, enabling a more thorough exploration of the solution space becomes particularly valuable.

Enhancement 4. We introduce MFSPart-Ensemble, an enhanced version of MFSPart that improves solution quality by integrating solution ensembling into the uncoarsening process. Unlike simple local search, this approach generates multiple initial solutions on the coarsest graph and progressively merges their promising structures. This collaborative search strategy effectively preserves high-quality structures across different coarsening levels, yielding a superior final solution albeit at the cost of increased runtime.

D. Our Contribution

Our key contribution in this paper is summarized as below:

1) **Problem formulation improvement:** We propose a more comprehensive and flexible formulation for multi-FPGA system partitioning to enhance algorithm adaptability. Our approach generalizes the existing "one-hop constraint" to a maximum hop-count threshold, accounts for FPGA capacity, supports both fixed and non-fixed node settings, and jointly optimizes the *driver–sink pair cut size*, the *connectivity metric*, and the *mean hop count* to reduce latency, save I/O resources, and simplify routing.
2) **A new framework and its extension:** We identify inefficiencies in performing propagation before coarsening, which is common in existing works. To address this, we propose a new framework, MFSPart, that removes pre-coarsening propagation and performs it solely on the

[4]Throughout this paper, we use *violation* to generally refer to the case of violating the "low-hop constraint".

[5]In this paper, a *violating node pair* refers to two nodes assigned to different FPGAs whose distance exceeds $H_{\max}$, thereby violating the "low-hop constraint".

[6]For example, they can only shorten the distance between node pairs from 2–5 to 0 or 1 to meet the "one-hop constraint" ($H_{\max} = 1$), but are unable to reduce a distance-5 node pair to distance 2, which could have facilitated further improvements on finer graphs.

coarsest graph. We also revise strategies in coarsening, initial partitioning, and refinement stages, yielding notable improvements in runtime and solution quality. To further mitigate local optima in the refinement phase, we extend MFSPart to MFSPart-Ensemble, which integrates a solution ensembling technique within the uncoarsening and refinement process, achieving consistently better solutions at the cost of increased runtime.

3) **Comprehensive experimental design:** We conduct thorough experiments across various scenarios, including different FPGA sizes, different maximum hop-count thresholds, presence or absence of fixed-node constraints, and various benchmark scales. These settings comprehensively cover those used in many existing works. We also provided comprehensive further analyses, such as parameter sensitivity analysis and comparisons in extended scenarios. To ensure a fair comparison, our algorithms are evaluated against each baseline within its own applicable experimental setting (see Table I). The results demonstrate that our methods are broadly applicable and consistently identify high-quality, feasible solutions, achieving state-of-the-art performance in all evaluated metrics.

II. Problem Formulation

Given a netlist represented by a hypergraph $H(V, E_H)$, where V is the set of nodes (cells) and E_H is the set of hyperedges (nets), the goal is to partition the netlist into multiple parts, with each part assigned to a corresponding FPGA in the multi-FPGA system $\hat{G}(\hat{V}, \hat{E})$, where $\hat{V}$ denotes the set of FPGAs and $\hat{E}$ represents their interconnections. To achieve a high-quality partition, our work adopts a multi-objective optimization approach that simultaneously considers the following three metrics:

1) **Driver-sink pair cut size:** Each n-pin net, represented by a hyperedge $e_H \in E_H$ (with one driver and $n-1$ sinks), is decomposed into a set of $n-1$ two-pin nets (edges), where each edge inherits the weight of the original hyperedge[7]. For a given hyperedge e_H, we denote this set of derived edges as $E(e_H)$. The set of all two-pin nets for the entire hypergraph is therefore $E = \bigcup_{e_H \in E_H} E(e_H)$. The *driver-sink pair cut size* C_{ds} is the total weight of all derived driver-sink edges that are cut across FPGAs, reflecting the overall system latency:

$$C_{\text{ds}} = \sum_{e_H \in E_H} \sum_{\substack{(u,v) \in E(e_H) \\ \hat{u} \neq \hat{v}}} w(u, v) \tag{1}$$

where $\hat{u}$ and $\hat{v}$ denote the FPGAs to which nodes u and v are assigned, respectively. The term $w(u, v)$ represents the aggregated weight of the driver-sink pair (u, v). Since multiple hyperedges may decompose into edges connecting the same pair of nodes, their inherited weights are summed to determine $w(u, v)$.

2) **Connectivity metric:** Unlike the *driver-sink pair cut size*, this metric is calculated based on the original hyperedges.

$$C_{\text{con}} = \sum_{e_H \in E_H} w(e_H) \cdot |S(e_H) \setminus D(e_H)| \tag{2}$$

[7]By default, all nodes and hyperedges are initialized with unit weights.

TABLE II: Summary of key notations used in this manuscript.

Term	Description
$H(V, E_H)$	The netlist graph. V represents the set of nodes, and E_H represents the set of hyperedges.
E	The set of all 2-pin nets (driver-sink pairs) obtained from the decomposition of the hyperedges.
$\hat{G}(\hat{V}, \hat{E})$	The MFS graph. $\hat{V}$ represents the set of FPGAs, and $\hat{E}$ represents the set of their interconnections.
e_H	A hyperedge in set E_H representing an n-pin net.
e	An edge in set E representing an 2-pin net (a driver-sink pair).
k	The number of FPGA in $\hat{G}$, i.e., $k = \lvert\hat{V}\rvert$.
$w(u, v)$	The edge weight between nodes u and v.
$w(e_H)$	The weight of a hyperedge e_H.
$w(u), w(v)$	The node weights of nodes u and v.
Q	The fixed node set.
$\hat{u}$	The FPGA of node $u \in Q$.
$\text{CF}(u)$	The candidate FPGAs of node u.
$H_{\max}$	The maximum hop-count threshold.

where $w(e_H)$ denotes the hyperedge weight, and $S(e_H)$ and $D(e_H)$ represent the sets of FPGAs containing the sink nodes and driver node of e_H, respectively. $|S(e_H) \setminus D(e_H)|$ thus counts FPGAs with sinks but without the driver, encouraging assigning all net nodes onto a single FPGA to reduce I/O overhead and routing complexity.

3) **Mean hop count:** Within the maximum hop-count threshold, we aim to further reduce the average number of hops between each driver-sink pair, thereby improving signal propagation efficiency.

The objective is to find a partition P that minimizes the above metrics while meeting the following constraints:

1) **Low-hop constraint:** For each driver-sink pair (u, v), the hop count between their assigned FPGAs, denoted as $d(\hat{u}, \hat{v})$, must not exceed a user-defined threshold $H_{\max}$ (i.e., $d(\hat{u}, \hat{v}) \leq H_{\max}$).
2) **FPGA capacity constraint:** The sum of the weight of nodes assigned to each FPGA α must not exceed its capacity $C(\alpha)$.

$$\sum_{u \in \alpha} w(u) \leq C(\alpha) \tag{3}$$

where $w(u)$ is the weight of the node u.
3) **Fixed-node constraint (optional):** Each initial fixed node must be assigned to its designated FPGA.

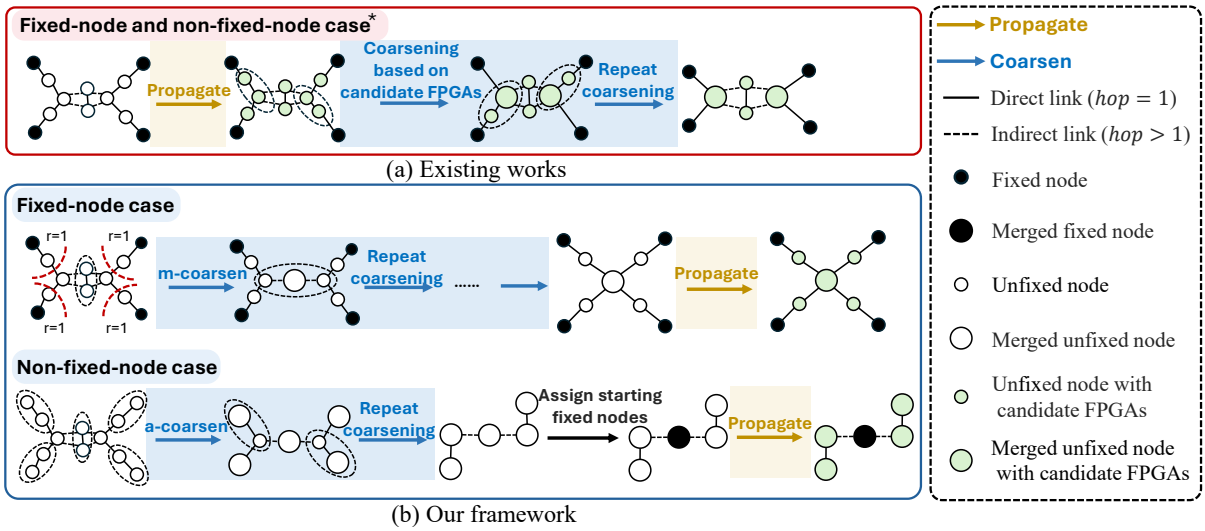
Table II lists the key notations used throughout this work.

III. MFSPart: Decoupling Coarsening from Propagation for Flexible MFS Partitioning

To solve the multi-objective MFS partitioning problem, we introduce MFSPart, a new multi-level framework comprising three stages: coarsening (Sec. III-A), propagation & initial partitioning (Sec. III-B), and uncoarsening & refinement (Sec. III-C).

A fundamental distinction between MFSPart and conventional methods lies in the execution order of propagation and coarsening. As illustrated in Fig. 4(a), conventional methods [12], [21], [25], [26] typically perform propagation *before* coarsening; this order introduces three major limitations:

1) **Risk of Violations:** Candidate FPGA information evolves as the hypergraph is coarsened. Relying on premature



* The non-fixed-node case in existing works selects starting fixed nodes to execute this process.

Fig. 4: Our framework postpones propagation until after coarsening. A comparison between (a) existing works that propagate on the original graph before coarsening, and (b) our framework, which coarsens first and propagates only on the coarsest graph.

propagation results before coarsening leads to outdated information, potentially leading to violations.

2) **High Computational Cost:** Executing propagation on the large-scale original hypergraph is time-consuming.
3) **Constrained Topology Preservation:** Since coarsening is constrained by candidate FPGAs, its ability to focus on capturing the global topology structure is restricted.

The cornerstone of MFSPart is to address these issues by reversing the order of coarsening and propagation, as illustrated in Fig. 4(b). Our key insight is that propagation can be executed on any coarsening level to compute the candidate FPGAs for the nodes at that level. Performing propagation on the original hypergraph—as done in prior work—causes the candidate-FPGA information to evolve and quickly become invalid as nodes are merged, which directly leads to inaccurate candidate sets for clusters. By postponing propagation until the coarsest hypergraph is formed, we guarantee that every coarse node (i.e., cluster) receives an accurate candidate FPGA set, while also eliminating the runtime bottleneck of propagation on the original hypergraph. Moreover, our coarsening can focus solely on the topology of the hypergraph, which facilitates constructing higher-quality coarsening hierarchies. Taken together, this paradigm shift offers a more rational, robust, and efficient approach to guiding multi-FPGA system partitioning under topology constraints.

Beyond this, MFSPart integrates novel strategies across all phases to optimize the target metrics. Finally, in Sec. IV, we will introduce MFSPart-Ensemble, a variant that integrates a solution ensembling technique within the uncoarsening and refinement process to further enhance partitioning quality.

A. Step1: Coarsening

The purpose of coarsening is to merge nodes that are more likely to be assigned to the same FPGA, thereby reducing the graph's size and simplifying the problem. As previously discussed, in existing works, performing propagation prior to coarsening introduces several limitations.

In contrast, our coarsening operates without relying on any candidate FPGA information. Instead, it focuses solely on nodal affinity based on the driver–sink edge decomposition described in Sec. II. This transformation is particularly effective for pursuing low-hop assignments for driver-sink pairs and significantly reduces the complexity. Formally, we define the affinity between nodes u and v as:

$$\text{affinity}(u,v) = \frac{w(u,v)}{w(u)w(v)} \tag{4}$$

In each coarsening iteration, we iteratively merge node pairs with the highest affinity, breaking ties randomly. Here, the weight of a merged node is the sum of its constituent nodes, and the edge weight between two merged nodes is calculated as the total weight of all connections between them. **(1) In the non-fixed-node case**, we simply follow this basic operation, termed *a-coarsen* (affinity-coarsen). **(2) In the fixed-node case**, the aforementioned coarsening may lead to violations due to the initial fixed nodes. To mitigate this and provide greater flexibility in initial partitioning, we extend the affinity-based coarsening by introducing a margin in two ways: 1) Nodes within r hops of initial fixed nodes are prevented from merging throughout the coarsening phase. 2) Merging is prohibited if it would reduce the distance between initial fixed nodes[8] to less than the distance between their FPGAs plus a constant margin m—this is termed *m-coarsen* (margin-coarsen). This essentially relaxes the "low-hop constraint" imposed by initial fixed nodes on the coarsest graph. These two coarsening strategies are depicted in Fig. 4(b).

In addition, to support the optimization of the *connectivity metric*, we explicitly track the transformation of each hyperedge throughout the coarsening process. After each coarsening iteration, two or more hyperedges are considered identical—and their weights combined—if they share the same driver node and an identical set of sink nodes. This procedure

[8]To improve efficiency, we merge initial fixed nodes assigned to the same FPGA before coarsening.

ensures that the measurement of the *connectivity metric* is kept accurate before and after coarsening, while simultaneously simplifying the hypergraph representation.

Advantage. Our proposed coarsening strategy effectively addresses three limitations of existing frameworks. First, conducting propagation on the coarsest hypergraph provides accurate candidate FPGA information for initial partitioning. Second, it eliminates the time-consuming propagation across the original graph. Finally, it preserves the original graph's topology effectively using a-coarsen and m-coarsen.

B. Step2: Propagation & Initial Partitioning

In this stage, each unfixed node is assigned to one of the FPGAs, aiming to generate a high-quality initial solution for the refinement stage.

For the propagation, we employ the efficient algorithm presented in [25]. **In the fixed-node case**, the FPGA information of initial fixed nodes is preserved throughout coarsening, thereby directly enabling propagation on the coarsest graph. **In the non-fixed-node case**, we select the node with the maximum normalized degree, defined as the degree divided by its node weight, and assign it to the FPGA with the highest degree, serving as the start node for propagation. In contrast to [21], which pre-assigns fixed nodes using the single-source shortest path algorithm before coarsening, our approach avoids such early constraints, thereby offering greater flexibility for the initial partitioning.

For unassigned nodes, existing methods [12], [21], [25] tend to overpopulate single FPGAs, restricting solution space exploration during refinement. Instead, we employ a two-phase assignment strategy. (1) The first phase uses a probabilistic approach for nodes with **non-empty candidate FPGA sets** to promote diversity, (2) while the second phase applies a deterministic method for nodes with **empty candidate sets** to mitigate violations. Guided by a comprehensive consideration of capacity, cut size, violations, and FPGA degree, this hybrid approach yields a higher-quality initial solution.

Phase ❶: Probabilistic Assignment for Nodes with Candidate FPGAs. First, we iteratively process unassigned nodes with candidate FPGAs, prioritizing the node u with the highest priority $P_{\text{ini}}(u)$ determined by its connectivity to fixed neighbors. Once the node u is selected, we assign it to a candidate FPGA using a probabilistic approach. The probability of assigning u to FPGA α, denoted as $\Pr(u, \alpha)$, is calculated based on the set of assignment scores $\{S_{\text{ini}}(u, \beta)\}$ in Eq. 6 for node u across all its candidate FPGAs β. This distribution prioritizes FPGAs with larger available capacity, lower cut size and higher degrees while retaining small probabilities for other candidates, thereby preserving strategic randomness for focused solution space exploration.

Specifically, we determine the assignment priority $P_{\text{ini}}(u)$ of each unassigned node u as follows:

$$P_{\text{ini}}(u) = \frac{\sum_{(u,v)\in E, v\in Q} w(u,v)}{\sum_{(u,v)\in E} w(u,v)} \tag{5}$$

where Q denotes the fixed node set, i.e., nodes that have been assigned to specific FPGAs. We define the assignment score for assigning a node u to its candidate FPGA α as follows:

$$S_{\text{ini}}(u,\alpha) = \sum_{\substack{(u,v)\in E \\ \text{FPGA}(v)=\alpha}} w(u,v) - \frac{\theta}{\text{capacity}(\alpha)} + \eta \cdot d(\alpha) \tag{6}$$

where θ and η are pre-defined parameters, capacity(α) is the remaining capacity of FPGA α after we assign node u onto FPGA α, $d(\alpha)$ is the degree of FPGA α. The probability of assigning node u to FPGA α is then calculated as below:

$$\Pr(u,\alpha) = \frac{e^{(S_{\text{ini}}(u,\alpha))/T_u}}{\sum_{\beta\in \text{CF}(u)} e^{(S_{\text{ini}}(u,\beta))/T_u}} \tag{7}$$

where CF(u) is the set of candidate FPGAs for node u. We subtract the maximum score $S_{\text{ini}}(u,\beta)$ among all candidates before exponentiation to improve numerical stability and avoid underflow. We also introduce a temperature coefficient T_u to control the probability distribution's sharpness.

If assigning node u to one of its candidate FPGAs would cause the candidate FPGA set of other nodes to become empty, we remove this FPGA from CF(u) and assign u to another FPGA based on the probabilistic distribution. After assigning node u to its designated FPGA, we update the candidate FPGA sets for the remaining nodes using propagation and adjust the priorities of u's neighbors. In this process, there are some nodes whose candidate sets become empty, which are added to the violation set R for further processing.

Phase ❷: Deterministic Assignment for Nodes without Candidates. Once all feasible nodes have been assigned, we address the nodes in R by assigning them to any suitable FPGA with sufficient capacity. Specifically, we calculate the assignment score $S_{\text{ini}}(u,\alpha)$ for assigning node $u \in R$ into any FPGA α as below:

$$\begin{aligned} S_{\text{ini}}(u,\alpha) = &\sum_{\substack{(u,v)\in E \\ \text{FPGA}(v)=\alpha}} w(u,v) - \frac{\theta}{\text{capacity}(\alpha)} + \eta \cdot d(\alpha) \\ &- \lambda \cdot \sum_{\substack{(u,v)\in E \\ dist:=dis(\text{FPGA}(v),\alpha) > H_{\max}}} w(u,v)(1+\mu\cdot(dist - H_{\max})) \end{aligned} \tag{8}$$

where θ, η, λ and μ are pre-defined parameters. Building upon Eq. 6, this formula introduces the final term to handle potential violations, encouraging solutions to minimize the distance between violated node pairs.

In contrast to the probabilistic approach, we assign violated nodes ($u \in R$) deterministically to minimize violation. Therefore, for each node $u \in R$, we compute its score $S_{\text{ini}}(u,\beta)$ for every FPGA β. We then use a priority queue to iteratively select the node-FPGA pair (u,β) with the highest score $S_{\text{ini}}(u,\beta)$, assign node u to FPGA β, and update the scores $S_{\text{ini}}(v,\cdot)$ for all neighbors v of u.

Advantage. Our initial partitioning method employs a two-phase strategy, combining probabilistic and deterministic assignment. This approach comprehensively considers capacity, cut size, violations, and FPGA degrees to generate a high-quality initial solution. This facilitates a thorough exploration of the solution space in the uncoarsening & refinement phase.

C. Step3: Uncoarsening & Refinement

During uncoarsening and refinement, the merged nodes from the coarser graph are restored to the finer graph, and their FPGA assignments are refined using a customized FM algorithm to minimize cut size and resolve violations at that level. Specifically, FM computes the gain for all feasible moves, and iteratively selects the move with the highest gain, applying moves until the total gain reaches its maximum.

In the following, we detail our contributions to this phase from two key aspects: improvements to the FM gain function and acceleration strategies for the refinement process.

❶: FM gain function. The core of our refinement process is a carefully designed gain function G_{re} that guides node movements. The gain of moving a node u from its current FPGA to a target FPGA α is calculated as the difference between its compatibility scores S_{re} with respect to the target and current FPGAs:

$$G_{\text{re}}(u, \alpha) = S_{\text{re}}(u, \alpha) - S_{\text{re}}(u, \text{FPGA}(u)) \tag{9}$$

The compatibility score S_{re} (Eq. 10) balances three key objectives:

1) The first term targets the weighted total hop count of node u, simultaneously optimizing both the *mean hop count* and the *driver-sink pair cut size*.
2) The second term optimizes the *connectivity metric* to save I/O resources by minimizing the number of FPGAs spanned by each hyperedge.
3) The third term explicitly handles the distance of violating node pairs. Unlike the previous work [25] that accounts only for the existence of violations, our approach prioritizes moves that bring violating pairs closer, effectively reducing the severity of violations.

The compatibility score $S_{\text{re}}(u, \alpha)$ is formally defined as below, provided that the capacity of FPGA α suffices[9]:

$$\begin{aligned} S_{\text{re}}(u, \alpha) = & \sum_{i=0}^{H_{\max}} \sum_{\substack{(u,v)\in E \\ dis(\text{FPGA}(v),\alpha)=i}} (H_{\max} - i) \cdot w(u, v) \\ & - \gamma \cdot \sum_{e_H \ni u} C(e_H, \alpha) \cdot w(e_H) \\ & - \lambda \cdot \sum_{\substack{(u,v)\in E \\ dis(\text{FPGA}(v),\alpha)>H_{\max}}} w(u, v)(1 + \mu \cdot (dist - H_{\max})) \end{aligned} \tag{10}$$

where $C(e_H, \alpha)$ denotes the number of FPGAs spanned by hyperedge e_H, under the condition that node u is assigned in FPGA α while all other nodes in e_H remain at their current locations. The term $w(e_H)$ represents the weight of hyperedge e_H. The parameters γ, λ and μ are pre-defined.

❷: Acceleration strategies. While our comprehensive gain function improves partitioning quality, the time consumption can be a significant performance bottleneck. In [21], [25], FM dominates the overall runtime, accounting for 60–80% of the total execution time. To address this, we introduce two acceleration strategies.

1) Our observation suggests that during the FM process, nodes tend to be moved to nearby FPGAs: moves with distance $d = 1$ are prevalent, $d = 2$ less frequent, and $d = 3$ almost nonexistent. This makes sense because, in a generally good solution, a node and its neighbors should be assigned on FPGAs that are close to one another. Moving it a long distance usually causes more violations, as well as an increase in hop count. Therefore, it is unnecessary to consider moving nodes to distant FPGAs; moving to nearby FPGAs suffices.

2) Besides, we observed that when a certain proportion of node moves no longer improves the total gain, the FM process at this coarsening level has nearly reached its maximum total gain. Thus, we employ the "early stopping" strategy, terminating FM at this level once the proportion of ineffective moves exceeds a threshold, which is also employed in hMETIS [7].

Based on these insights, our acceleration strategies are:

1) Restricting node moves to FPGAs within a certain distance d.
2) An "early stopping" strategy that halts FM once ineffective moves exceed a threshold.

Advantage. Our enhanced FM algorithm's gain function is carefully designed to co-optimize the low-hop requirement and the *connectivity metric*, improving overall system latency and I/O resource utilization. In addition, our method accelerates computation by limiting the node movement distance and employing an "early-stopping" strategy.

IV. MFSPart-Ensemble: Integrating Cut Overlay Clustering into Uncoarsening

The iterative nature of the uncoarsening and refinement phase in conventional multi-level partitioners often confines the search to a narrow path of incremental local improvements, struggling to escape suboptimal solutions. To overcome this fundamental limitation, we introduce a recombination strategy inspired by a solution ensembling technique, *cut overlay clustering* [9]. The core idea, illustrated in Fig. 5(a) for the case of two solutions ($c = 2$), is to build a clustered hypergraph by merging nodes in the same partition across all parent solutions $(V_1, \ldots, V_c)$, and then identify an improved solution from this clustered hypergraph. This ensemble-based mechanism allows for a powerful exploration beyond the confines of a single solution trajectory by integrating stable structures from multiple high-quality partitions.

While prior works [9], [33] employed this technique as a post-processing step on the final graph, we posit that its power is more profound when integrated *within* the multi-level process itself. By applying it at various uncoarsening stages (Fig. 6), we transform the refinement from a simple local search into a sophisticated, collaborative search. Accordingly, we propose MFSPart-Ensemble (Alg. 1), an advanced framework that embeds this ensembling core into our multi-level partitioner. The overall framework is summarized as follows:

1) We first initialize $s(= 5)$ solutions on the coarsest graph. Each i-th solution is combined with the $(i + 1)$-th (the 5-th combined with the 1st) via *cut overlay clustering*.

[9]If an FPGA has insufficient capacity for node u, we need not consider moving node u there; thus, defining the compatibility score is not required.

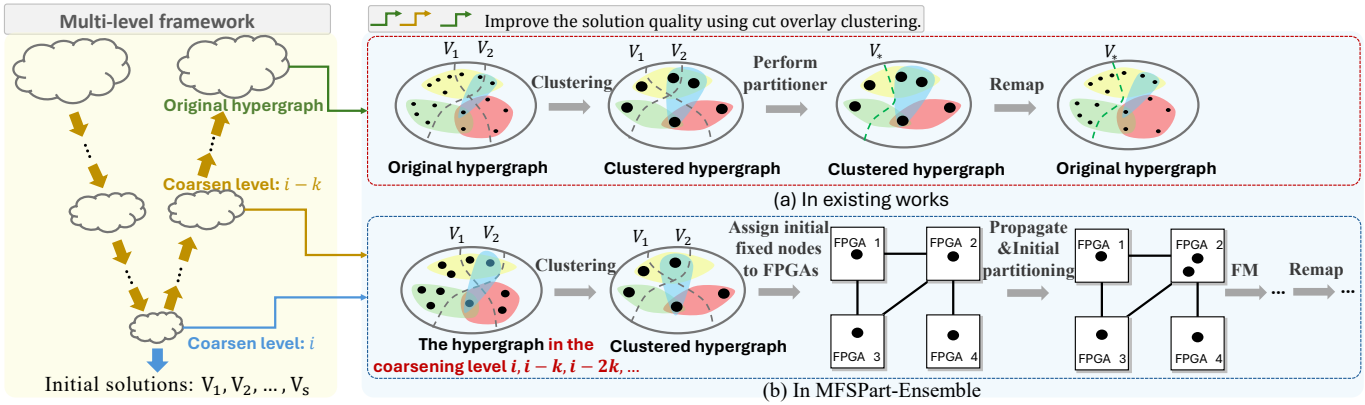


Fig. 5: Comparison of *cut overlay clustering* usage: (a) standard hypergraph partitioners [9], [33]; (b) our MFSPart-Ensemble.

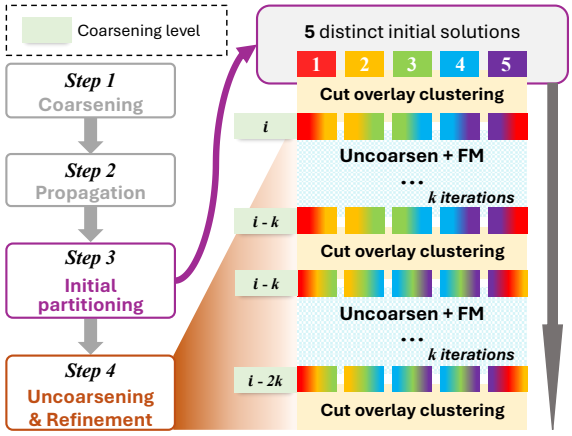


Fig. 6: MFSPart-Ensemble uses 5 colors to represent initial partitioning schemes. Adjacent solutions are combined to produce an improved solution using cut overlay clustering.

This produces a set of *clustered hypergraphs*, which are more condensed than the coarsest graph, each inheriting beneficial traits from two parents and serving as the foundation for generating an improved offspring solution.

2) Then, after every $k(=6)$ iterations of the uncoarsening phase, we apply the same generation process in step 1 again to produce 5 new offspring.

As detailed in lines 15–22 of Alg. 1, solutions are projected to the finer level and refined via FM in each iteration. To enhance solution quality through the ensemble technique, *cut overlay clustering* is selectively applied only when $i \equiv \alpha \pmod{k}$. The rationale for not setting $k = 1$ is to allow solutions to develop distinct characteristics through multiple uncoarsening rounds while reducing runtime overhead.

The clustering-based generation process is detailed in Fig. 5(b). After constructing the clustered hypergraph, the resulting nodes must be assigned to FPGAs to form a new solution. We reuse the probabilistic assignment strategy from the initial partitioning stage (Sec. III-B), which requires initial fixed nodes to guide propagation. The selection method differs by setting: in the fixed-node case, we directly identify nodes formed by the initial fixed nodes on the clustered hypergraph and assign them to their corresponding FPGAs; in the non-fixed-node case, we identify clustered nodes consistently assigned to the same FPGA across solution pairs and fix them as anchors for propagation[10]. When no consistently assigned nodes exist, we randomly fix a node to the FPGA with the most interconnections. After the assignment, a local search (FM) refines the solution. The solution on the clustered hypergraph is then remapped to the corresponding coarsening level of the hypergraph before clustering.

Algorithm 1 MFSPart-Ensemble

Require: Netlist $G(V, E)$, FPGA system $\hat{G}(\hat{V}, \hat{E})$, optional fixed nodes N_{fixed}
Ensure: Partition result P_{best}

1: $G_0 \leftarrow G$, $i \leftarrow 0$
2: **repeat**
3: $\quad G_{i+1} \leftarrow \textsc{Coarsen}(G_i)$; $i \leftarrow i+1$ ▷ Sec. III-A
4: **until** $|V_{i-1}| - |V_i| \leq \tau$
5: **if** $N_{\text{fixed}} = \emptyset$ **then**
6: $\quad N_{\text{fixed}} \leftarrow \{\arg\max_{v \in V}(\text{norm\_deg}(v))\}$
7: $CF \leftarrow \textsc{Propagate}(G_i, N_{\text{fixed}}, \hat{G})$ ▷ Sec. III-B
8: $pop_i \leftarrow \textsc{InitialPartitioning}(G_i, \hat{G}, CF)$
9: $pop_i \leftarrow \textsc{CutOverlay}^{\star}(pop_i, G_i, \hat{G})$ ▷ Sec. IV
10: Apply FM to each $pop_{i,t}$ for each $t \in [1, s]$ ▷ Sec. III-C
11: $\alpha \leftarrow i \bmod k$
12: **while** $i > 0$ **do**
13: $\quad i \leftarrow i-1$; $pop_i \leftarrow \textsc{Project}(G_i, G_{i+1}, pop_{i+1})$
14: $\quad$ **if** $i \equiv \alpha \pmod{k}$ **then**
15: $\quad\quad pop_i \leftarrow \textsc{CutOverlay}(pop_i, G_i, \hat{G})$
16: $\quad$ Apply FM to each $pop_{i,t}$ for each $t \in [1, s]$
17: $P_{\text{best}} \leftarrow \textsc{SelectBest}(G_0, \hat{G}, pop_0)$
18: **return** P_{best}

* CUTOVERLAY represents the operations in Fig. 5(b), including cut overlay clustering, propagation, initial partitioning, FM, and so on.

Advantage. Our enhanced variant, MFSPart-Ensemble, pioneeringly enables solution learning and recombination during uncoarsening, achieving more thorough exploration of the solution space and increasing the likelihood of discovering superior solutions.

[10]To preserve diversity, we limit anchors to at most half of the clustered hypergraph nodes; if this threshold is exceeded, we randomly sample from the identified set.

V. Experiments

A. Experimental Configuration

For experiments, we utilize benchmark netlists from the Titan23 suite [34] and 8 generated datasets from [12], as listed in Table III. These netlists are preprocessed by removing isolated nodes, as such nodes can be freely assigned to any FPGA. We also employ two multi-FPGA system configurations derived from the ICCAD 2019 Contest B [35]: MFS1, consisting of 8 FPGAs and 11 interconnections, and MFS2, comprising 43 FPGAs and 214 interconnections. The selection of these datasets and FPGA configurations aligns with prior studies [12], [25], [26] for fair comparison.

Unlike previous works assuming unlimited FPGA capacity, we set explicit FPGA capacities, supporting different levels of resource constraints. For example, in Table IV and VIII, '$5C$' and '$2C$' represent capacities set at two and five times the average vertex count per FPGA, respectively, calculated as $5 \times \frac{|V|}{k}$ (as more relaxed constraint) and $2 \times \frac{|V|}{k}$ (as tighter constraint). Besides, we divide the experiments into four groups based on the presence or absence of initial fixed nodes and the value of the maximum hop-count threshold H_{max} to test the efficiency of the algorithm under different experimental scenarios:

Setup ❶ With initial fixed nodes and H_{max}=1.
Setup ❷ With initial fixed nodes and H_{max}=2.
Setup ❸ Without initial fixed nodes and H_{max}=1.
Setup ❹ Without initial fixed nodes and H_{max}=2.

The experimental setup details are given below:

- In our experiments, we consistently adopted the following default parameter settings for all benchmarks without instance-specific tuning: For m-coarsen in the fixed-node case, the default radius r is set to 1 and the margin m is set to 3. In the initial partitioning phase, the assignment score for nodes with non-empty candidate FPGA sets (Eq. 6) uses $\theta = 10^4$ and $\eta = 500$ as the default value. The same θ and η also appear in the score for nodes with empty-candidate FPGA sets (Eq. 8), which further incorporates $\lambda = 10^4$ and $\mu = \frac{1}{10}$; in Eq. 10, the same values are used for λ and μ, and the parameter γ is set to 15. The temperature coefficient T_u (Eq. 7) is set to $\sum_{(u,v)\in E, v\in Q} w(u,v)$. In the refinement stage, the distance limit d for node movement is set to 2, the threshold for the number of ineffective moves for "early stopping" is set as $0.15n$ for the fixed-node case and $0.2n$ for the non-fixed-node case. The coarsening termination threshold in Alg. 1 is set to $\tau = 30$. All common parameters remain consistent across both MFSPart and MFSPart-Ensemble.
- The fixed-node configurations on the Titan23 benchmark and the 8 generated datasets (`case1`–`case8`) are consistent with those used in [12], [25].
- A solution is marked "Fail" if no 0-violation solution is found.
- In Tables IV to XI, where 'F' and 'N' denote the fixed-node case (i.e., with initial fixed nodes) and the non-fixed-node case respectively, we identify each design by its index number, as defined in Table III, and we adopt baselines applicable to the same scenario as those in their papers, as specified in Table I in our introduction. To provide a thorough comparison among different methods, we report four metrics: the *driver–sink pair cut size* (Cut), the *connectivity metric* (Con), the *mean hop count* per driver–sink pair (Hop), and the runtime (Time). The Cut and Con metrics are measured in thousands (K), the Time is measured in seconds (s), and the Hop metric is reported in multiples of 10^{-3}. Furthermore, we provide a normalized average (Norm. Avg.) for each metric, calculated as the geometric mean relative to MFSPart. For each method, the normalized average is calculated only over the designs that it processes successfully.
- We bold the result of MFSPart if it achieves the best performance against all baselines [12], [21], [25], [26], and highlight the result of MFSPart-Ensemble in green and bold if it is the best performer overall.
- All experiments were conducted using C++17 on a single core of an Intel(R) Xeon(R) Platinum 8375C CPU @ 2.90GHz, running Ubuntu 20.04 with 512GB of memory.

TABLE III: Benchmarks description.

Titan23 benchmark [34]							
#	**dataset**	**# nodes**	**# nets**	**#**	**dataset**	**# nodes**	**# nets**
1	sparcT1_core	92K	93K	12	minres	261K	321K
2	neuron	92K	125K	13	cholesky_bdti	266K	343K
3	stereo_vision	94K	127K	14	denoise	276K	357K
4	des90	111K	140K	15	sparcT2_core	300K	303K
5	SLAM_spheric	113K	142K	16	gsm_switch	493K	508K
6	cholesky_mc	113K	145K	17	mes_noc	548K	578K
7	segmentation	138K	179K	18	LU_Network	635K	727K
8	bitonic_mesh	192K	235K	19	LU230	574K	669K
9	dart	202K	223K	20	sparcT1_chip2	821K	821K
10	openCV	217K	284K	21	directrf	931K	1,375K
11	stap_qrd	240K	290K	22	bitcoin_miner	1,089K	1,448K
Synthetic benchmark from [12]							
#	**dataset**	**# nodes**	**# nets**	**#**	**dataset**	**# nodes**	**# nets**
1	case1	300K	749K	5	case5	700K	1,748K
2	case2	400K	999K	6	case6	800K	1,998K
3	case3	500K	1,249K	7	case7	900K	2,248K
4	case4	600K	1,498K	8	case8	1,000K	2,498K

In Section V-B, we comprehensively compare our methods against baselines [12], [21], [25], [26] under various settings, followed by further analysis in Section V-C.

B. Comprehensive Analysis

Setup ❶ With initial fixed nodes, H_{max}=1.

In Tables IV and V, we report results for the Titan23 benchmark on MFS1 and the synthetic benchmark on MFS2, with FPGA capacity limits of '$5C$' and '$35C$', respectively. The results first highlight the superiority of our framework in finding feasible solutions: under the strict one-hop constraint, both MFSPart and MFSPart-Ensemble find feasible solutions in all cases, whereas the baselines frequently fail on MFS2 and occasionally on MFS1. Beyond feasibility, our methods also perform strongly on all three optimization objectives. Taking MFSPart as the reference (1.00), the normalized averages (Norm. Avg.) for the best baseline, MFSPart, and MFSPart-Ensemble are:

TABLE IV: Results on Titan23 (MFS1, 8 FPGAs) — **F**, $H_{\max}$ = 1, FPGA capacity = '5C'.

Design	[12] Cut (K)	[12] Con (K)	[12] Hop $\times10^{-3}$	[12] Time (s)	[25] Cut (K)	[25] Con (K)	[25] Hop $\times10^{-3}$	[25] Time (s)	[26] Cut (K)	[26] Con (K)	[26] Hop $\times10^{-3}$	[26] Time (s)	MFSPart Cut (K)	MFSPart Con (K)	MFSPart Hop $\times10^{-3}$	MFSPart Time (s)	MFSPart-Ensemble Cut (K)	MFSPart-Ensemble Con (K)	MFSPart-Ensemble Hop $\times10^{-3}$	MFSPart-Ensemble Time (s)
1	52	17.8	126	30	27	2.3	67	29	29	4.2	72	26	31	**2.3**	75	25	30	**1.5**	74	109
2	61	27.3	198	27	46	2.9	152	23	45	2.4	148	23	46	**1.0**	149	17	**45**	**1.0**	**148**	61
3	37	13.2	131	19	25	3.2	90	17	26	3.6	90	16	**23**	**0.7**	**83**	14	**23**	**0.5**	**82**	75
4	45	4.1	72	19	50	8.1	81	36	59	15.6	95	542	**41**	**1.5**	**70**	28	53	**0.9**	84	84
5	15	7.2	31	16	13	4.8	27	28	39	14.7	85	33	**10**	**1.4**	**21**	24	**10**	**1.4**	**21**	91
6	63	5.5	138	13	79	11.5	171	30	Fail				**62**	**2.6**	**135**	25	**59**	**1.9**	**129**	101
7	15	2.4	25	19	5	1.6	8	34	10	1.7	17	33	**3**	**0.5**	**5**	25	**2**	**0.4**	**3**	118
8	78	6.4	71	59	79	6.9	72	67	80	8.5	73	64	**74**	**1.7**	**67**	56	**74**	**1.2**	**67**	204
9	97	7.7	107	136	98	7.7	109	64	Fail				**95**	**1.8**	**106**	48	**95**	**1.6**	**105**	226
10	72	6.5	74	30	111	20.9	115	68	73	7.8	76	65	**72**	**2.1**	**75**	68	**71**	**2.0**	**74**	265
11	176	84.8	186	92	158	14.6	168	76	155	7.5	164	73	**142**	**2.0**	**150**	52	**142**	**1.9**	**150**	169
12	99	35.7	103	215	80	9.3	84	61	77	4.7	80	58	**77**	**2.7**	**80**	47	**75**	**2.0**	**79**	171
13	147	10.4	143	33	148	7.8	144	71	Fail				**145**	**3.6**	**142**	47	**145**	**2.5**	**141**	226
14	20	4.9	16	30	7	2.7	6	71	6	3.1	5	68	7	**0.6**	6	50	8	**0.6**	6	176
15	138	42.4	105	150	57	2.3	43	94	80	5.3	61	90	**54**	**2.3**	**41**	88	**53**	2.4	**40**	406
16	498	142.4	228	234	Fail				Fail				**319**	**19.0**	**146**	156	**166**	**2.3**	**76**	613
17	238	15.1	91	106	218	9.7	83	185	217	11.2	83	176	**199**	**2.8**	**76**	165	**197**	**1.5**	**75**	515
18	299	103.8	116	545	215	25.4	84	174	217	19.3	85	166	**186**	**4.4**	**72**	187	191	**3.0**	74	609
19	254	46.2	111	81	254	50.0	111	172	353	41.5	154	164	369	**15.1**	161	158	405	**7.5**	177	628
20	313	98.4	97	520	207	13.6	65	260	Fail				**193**	**2.3**	**61**	242	**185**	**2.3**	**58**	954
21	365	30.3	109	113	370	26.3	111	245	397	23.4	119	232	**353**	**2.9**	**106**	160	**352**	**1.9**	**106**	778
22	327	112.1	110	414	304	41.7	102	250	335	29.9	113	237	344	**3.6**	116	230	343	**3.5**	116	901
Norm Avg.	1.38	8.54	1.38	1.13	1.09	4.08	1.09	1.22	1.25	4.21	1.25	1.37	**1**	**1**	**1**	**1**	**0.96**	**0.72**	**0.96**	3.95

TABLE V: Results on the benchmark from [12] (MFS2, 43 FPGAs) — **F**, $H_{\max}$ = 1, FPGA capacity = '35C'.

Design	[12] Cut (K)	[12] Con (K)	[12] Hop $\times10^{-3}$	[12] Time (s)	[25] Cut (K)	[25] Con (K)	[25] Hop $\times10^{-3}$	[25] Time (s)	[26] Cut (K)	[26] Con (K)	[26] Hop $\times10^{-3}$	[26] Time (s)	MFSPart Cut (K)	MFSPart Con (K)	MFSPart Hop $\times10^{-3}$	MFSPart Time (s)	MFSPart-Ensemble Cut (K)	MFSPart-Ensemble Con (K)	MFSPart-Ensemble Hop $\times10^{-3}$	MFSPart-Ensemble Time (s)
1	Fail				Fail				Fail				**88**	**63**	**117**	273	**74**	**59**	**99**	1104
2	Fail				Fail				Fail				**82**	**67**	**82**	362	**75**	**61**	**75**	1543
3	Fail				Fail				167	124	134	862	**129**	**104**	**104**	476	**123**	**98**	**99**	2126
4	Fail				Fail				Fail				**163**	**123**	**109**	571	**124**	**100**	**83**	2519
5	Fail				Fail				Fail				**169**	**138**	**97**	698	**126**	**102**	**72**	2989
6	Fail				Fail				Fail				**187**	**149**	**94**	790	**178**	**144**	**89**	3407
7	Fail				Fail				Fail				**226**	**182**	**101**	901	**216**	**174**	**96**	3510
8	Fail				Fail				Fail				**293**	**221**	**117**	1044	**215**	**174**	**86**	4056
Norm Avg.	NA	NA	NA	NA	NA	NA	NA	NA	1.29	1.19	1.29	1.81	**1**	**1**	**1**	**1**	**0.85**	**0.88**	**0.85**	4.19

Best Baseline*: MFSPart : MFSPart-Ensemble =

$$\text{MFS1}\begin{cases}1.09:1.00:0.96 & \text{(Cut)}\\ 4.08:1.00:0.72 & \text{(Con)}\\ 1.09:1.00:0.96 & \text{(Hop)}\\ 1.22:1.00:3.95 & \text{(Time)}\end{cases}\quad\text{MFS2}\begin{cases}1.29:1.00:0.85 & \text{(Cut)}\\ 1.19:1.00:0.88 & \text{(Con)}\\ 1.29:1.00:0.85 & \text{(Hop)}\\ 1.81:1.00:4.19 & \text{(Time)}\end{cases}$$

* For a fair comparison, 'Best Baseline' refers to the best-performing method for each FPGA system: [25] for MFS1 and [26] for MFS2.

Overall, MFSPart substantially reduces the *connectivity metric* on MFS1, while simultaneously improving the *driver–sink pair cut size* and *mean hop count*, and also lowering runtime. MFSPart-Ensemble further improves these quality metrics over MFSPart, at the cost of additional runtime. On MFS2 (cases 1–8), the *connectivity metric* is slightly lower than the *driver–sink pair cut size*. This is expected because these datasets are dominated by 2-pin nets, where multiple nets sharing a common driver are treated as a single multi-pin net during connectivity evaluation to align with practical physical design constraints.

While MFSPart occasionally yields better results on specific metrics due to the stochastic nature of the refinement phase, MFSPart-Ensemble significantly enhances overall performance across the vast majority of datasets, albeit at the expense of higher runtime. This trade-off, observed consistently across setups 1–4, offers users flexibility: the standard MFSPart prioritizes computational efficiency, while MFSPart-Ensemble targets maximum solution quality. Since partitioning is an *offline* optimization step, the extra computation time is a worthwhile investment that significantly reduces I/O and latency during actual deployment. Furthermore, our experiments show that the overall effectiveness of MFSPart-Ensemble is more significant in complex multi-FPGA systems (MFS2) than in simpler ones (MFS1), a trend consistent across all configurations. This outcome highlights the limitations of the standard multi-level framework in handling complex FPGA topologies on its own. Concurrently, it underscores the value of MFSPart-Ensemble, by integrating the solution ensembling technique into the uncoarsening and refinement stage, we can uncover globally high-quality solutions more effectively, especially for complex FPGA systems.

Setup ❷ With initial fixed nodes, $H_{\max}$=2.

In this setup (Tables VI and VII), we extended $H_{\max}$ from 1 to 2 and adopted [26] as the baseline, which is applicable to this scenario. After increasing $H_{\max}$, all algorithms were able to find feasible solutions, indicating that the "one-hop constraint" is highly restrictive, while relaxing $H_{\max}$ significantly expands the feasible solution space. Relative to MFSPart (1.00), the normalized averages are:

Baseline [26]: MFSPart : MFSPart-Ensemble =

TABLE VI: Results on Titan23 (MFS1, 8 FPGAs) — **F**, $H_{\max}$ = 2, FPGA capacity = '$5C$'.

Design	[26] Cut (K)	[26] Con (K)	[26] Hop $\times10^{-3}$	[26] Time (s)	MFSPart Cut (K)	MFSPart Con (K)	MFSPart Hop $\times10^{-3}$	MFSPart Time (s)	MFSPart-Ensemble Cut (K)	MFSPart-Ensemble Con (K)	MFSPart-Ensemble Hop $\times10^{-3}$	MFSPart-Ensemble Time (s)
1	29	4.2	72	26	**24**	**1.5**	**58**	27	27	**1.2**	67	107
2	45	2.4	148	23	**45**	**1.0**	**146**	23	**45**	**0.9**	**146**	157
3	26	3.6	91	16	**22**	**0.6**	**76**	22	**20**	**0.3**	**72**	70
4	43	4.1	71	35	**41**	**1.5**	**67**	29	**40**	**1.0**	**65**	122
5	12	2.5	25	26	**9**	**1.7**	**19**	22	**8**	**1.6**	**18**	191
6	66	3.5	144	30	**58**	**1.6**	**127**	30	**58**	1.8	**125**	102
7	10	1.7	17	33	**2**	**0.3**	**3**	28	**1**	0.4	**2**	192
8	80	8.5	73	64	**72**	**1.7**	**65**	67	**72**	**1.4**	**65**	217
9	96	6.2	107	61	**84**	**2.1**	**93**	69	**82**	**1.8**	**91**	208
10	47	3.1	49	64	62	**1.7**	64	55	66	2.4	68	619
11	155	7.5	164	73	**142**	**3.6**	**150**	69	**132**	4.1	**140**	588
12	77	4.7	80	58	**76**	**2.1**	**79**	74	**75**	**1.6**	**79**	229
13	153	7.3	151	71	**139**	**4.4**	**136**	42	**138**	**3.7**	**134**	318
14	12	3.6	16	67	**7**	**0.4**	**5**	53	**6**	**0.4**	**5**	375
15	80	4.3	61	90	**45**	**3.3**	**34**	87	**41**	**2.1**	**32**	336
16	174	8.0	80	161	**156**	**8.4**	**71**	154	**145**	**4.5**	**66**	624
17	217	11.2	83	176	**193**	**3.1**	**74**	175	**188**	**3.0**	**72**	641
18	217	19.3	85	164	**190**	**3.3**	**74**	164	**181**	**3.2**	**71**	629
19	428	26.8	189	163	**370**	**16.4**	**161**	154	**347**	**16.4**	**151**	735
20	201	12.8	63	244	**164**	**7.6**	**51**	207	**138**	**1.9**	**43**	981
21	372	31.0	114	231	**360**	**5.0**	**108**	238	**359**	**4.4**	**108**	860
22	335	29.9	113	235	**333**	**3.3**	**112**	208	**331**	4.4	**111**	1163
Norm Avg.	1.24	2.88	1.28	1.05	**1**	**1**	**1**	**1**	**0.95**	**0.85**	**0.96**	4.85

TABLE VII: Results on the benchmark from [12] (MFS2, 43 FPGAs) — **F**, $H_{\max}$ = 2, FPGA capacity = '$35C$'.

Design	[26] Cut (K)	[26] Con (K)	[26] Hop $\times10^{-3}$	[26] Time (s)	MFSPart Cut (K)	MFSPart Con (K)	MFSPart Hop $\times10^{-3}$	MFSPart Time (s)	MFSPart-Ensemble Cut (K)	MFSPart-Ensemble Con (K)	MFSPart-Ensemble Hop $\times10^{-3}$	MFSPart-Ensemble Time (s)
1	64	50	164	421	**48**	**39**	**120**	255	**40**	**33**	**54**	1171
2	64	52	121	579	**62**	**50**	**120**	351	**50**	**41**	**50**	1604
3	102	82	153	743	**94**	**76**	**138**	447	**92**	**72**	**74**	2132
4	124	100	158	908	**118**	**96**	**117**	545	**117**	**95**	**79**	2616
5	136	109	100	1075	**116**	**96**	**92**	754	**110**	**88**	**63**	2905
6	145	118	90	1247	**130**	**105**	123	749	132	**105**	**66**	3465
7	194	156	115	1418	**177**	**144**	**107**	887	**175**	**143**	**78**	3414
8	164	134	76	1595	164	**134**	130	960	**142**	**116**	**57**	4138
Norm Avg.	1.10	1.09	1.00	1.62	**1**	**1**	**1**	**1**	**0.93**	**0.92**	**0.55**	4.41

$$\text{MFS1}\begin{cases}1.24:1.00:0.95 & \text{(Cut)}\\ 2.88:1.00:0.85 & \text{(Con)}\\ 1.28:1.00:0.96 & \text{(Hop)}\\ 1.05:1.00:4.85 & \text{(Time)}\end{cases}\quad \text{MFS2}\begin{cases}1.10:1.00:0.93 & \text{(Cut)}\\ 1.09:1.00:0.92 & \text{(Con)}\\ 1.00:1.00:0.55 & \text{(Hop)}\\ 1.62:1.00:4.41 & \text{(Time)}\end{cases}$$

The results under this setup reinforce the conclusions drawn from setup 1:

1) MFSPart consistently outperforms the baseline on all three objectives and reduces runtime.
2) MFSPart significantly optimizes the *connectivity metric*, leading to substantial savings in I/O resources for multi-FPGA systems.
3) MFSPart-Ensemble delivers further performance gains over MFSPart, though it requires additional computational overhead.
4) The effectiveness of MFSPart-Ensemble is more pronounced when applied to the complex MFS2 system.

Setup ❸ Without initial fixed nodes, $H_{\max}$=1.

In this setup (Tables VIII and IX), we consider the scenario without initial fixed nodes and set $H_{\max}$ to 1. Since there is no fixed-node constraint, we adopt a lower FPGA capacity to evaluate the performance of all algorithms.

In our experiments, while MFSPart and MFSPart-Ensemble consistently find feasible solutions in all test cases, the baseline [21] exhibits a relatively high failure rate (especially with a 100% failure rate on MFS2). We attribute this to the fact that it selects initial k fixed nodes via the single-source shortest path algorithm for propagation, which limits the flexibility of coarsening and the preservation of the hypergraph topology during this process. In contrast, our approach directly applies a-coarsen, thereby avoiding propagation on the original hypergraph. Furthermore, during initial partitioning, we assign the node with the maximum normalized degree onto FPGAs with the highest degree as the start node for propagation. This often ensures that the remaining nodes in the coarsest hypergraph can be assigned to FPGAs with minimal violations. Relative to MFSPart (1.00), the normalized averages are:

TABLE VIII: Results on Titan23 (MFS1, 8 FPGAs) — **N**, $H_{\max}$ = 1, FPGA capacity = '$2C$'.

Design	[21] Cut (K)	[21] Con (K)	[21] Hop $\times10^{-3}$	[21] Time (s)	MFSPart Cut (K)	MFSPart Con (K)	MFSPart Hop $\times10^{-3}$	MFSPart Time (s)	MFSPart-Ensemble Cut (K)	MFSPart-Ensemble Con (K)	MFSPart-Ensemble Hop $\times10^{-3}$	MFSPart-Ensemble Time (s)
1	Fail				**71**	**4.4**	**173**	29	**58**	5.5	**141**	109
2	97	6.3	315	21	**92**	**1.7**	**300**	20	**92**	**1.6**	**299**	104
3	56	5.3	200	17	**44**	**1.4**	**157**	15	45	**1.1**	160	46
4	94	12.8	152	36	**84**	**3.5**	**136**	34	**83**	**3.4**	**134**	115
5	Fail				**38**	**7.3**	**82**	33	**31**	**5.6**	**66**	125
6	Fail				**124**	**2.9**	**270**	24	**109**	**2.7**	**238**	142
7	16	5.2	26	35	**15**	**0.8**	**25**	27	**15**	**0.8**	**24**	91
8	151	12.1	137	66	**150**	**5.4**	**136**	67	**148**	**4.6**	**133**	349
9	Fail				**169**	**4.3**	**188**	59	175	**3.6**	194	318
10	141	10.5	146	66	**140**	**4.8**	**145**	64	141	**4.2**	147	181
11	Fail				**291**	**5.0**	**308**	57	298	**4.5**	316	176
12	Fail				**155**	**5.2**	**162**	51	**155**	**4.5**	**162**	154
13	301	16.4	293	66	**274**	**5.2**	**267**	62	**274**	**4.8**	**267**	304
14	39	11.2	31	72	**28**	**1.7**	**23**	58	**26**	**1.7**	**21**	273
15	Fail				**118**	**5.4**	**90**	113	**116**	**5.4**	**89**	351
16	Fail				**445**	**17.2**	**204**	196	**377**	**10.3**	**173**	516
17	538	71.1	206	181	**444**	**8.1**	**170**	184	**421**	10.0	**161**	460
18	Fail				**374**	**7.3**	**146**	187	**371**	**6.4**	**144**	475
19	Fail				**836**	**16.0**	**364**	173	**823**	**15.4**	**359**	434
20	436	35.5	136	266	**394**	**4.6**	**122**	265	**371**	**3.3**	**115**	509
21	Fail				**724**	**10.9**	**218**	188	**695**	**9.6**	**209**	517
22	Fail				**686**	**11.7**	**231**	258	**680**	11.8	**229**	637
Norm Avg.	1.12	4.33	1.12	1.08	**1**	**1**	**1**	**1**	**0.96**	**0.91**	**0.96**	3.39

TABLE IX: Results on the benchmark from [12] (MFS2, 43 FPGAs) — **N**, $H_{\max}$ = 1, FPGA capacity = '$8C$'.

Design	[21] Cut (K)	[21] Con (K)	[21] Hop $\times10^{-3}$	[21] Time (s)	MFSPart Cut (K)	MFSPart Con (K)	MFSPart Hop $\times10^{-3}$	MFSPart Time (s)	MFSPart-Ensemble Cut (K)	MFSPart-Ensemble Con (K)	MFSPart-Ensemble Hop $\times10^{-3}$	MFSPart-Ensemble Time (s)
1	Fail				**169**	**132**	**226**	403	**160**	**116**	**213**	2026
2	Fail				**240**	**179**	**240**	559	**205**	**148**	**205**	2789
3	Fail				**338**	**247**	**271**	717	**246**	**185**	**197**	3545
4	Fail				**456**	**316**	**305**	891	**291**	**220**	**194**	4384
5	Fail				**319**	**252**	**183**	1053	325	**249**	186	5160
6	Fail				**587**	**421**	**294**	1256	**417**	**308**	**209**	6131
7	Fail				**656**	**466**	**292**	1409	**392**	**305**	**174**	7102
8	Fail				**486**	**371**	**195**	1496	**440**	**336**	**176**	7824
Norm Avg.	NA	NA	NA	NA	**1**	**1**	**1**	**1**	**0.79**	**0.80**	**0.79**	4.99

Baseline [21]: MFSPart : MFSPart-Ensemble =

$$\text{MFS1}\begin{cases}1.12:1.00:0.96 & \text{(Cut)}\\ 4.33:1.00:0.91 & \text{(Con)}\\ 1.12:1.00:0.96 & \text{(Hop)}\\ 1.08:1.00:3.39 & \text{(Time)}\end{cases}\quad \text{MFS2}\begin{cases}\text{NA}:1.00:0.79 & \text{(Cut)}\\ \text{NA}:1.00:0.80 & \text{(Con)}\\ \text{NA}:1.00:0.79 & \text{(Hop)}\\ \text{NA}:1.00:4.99 & \text{(Time)}\end{cases}$$

The conclusion from setup 2 still holds true here.

Setup ❹ Without initial fixed nodes, $H_{\max}$=2.

In this setup (Tables X and XI), we extend the non-fixed-node case by increasing $H_{\max}$ from 1 to 2, under which all algorithms can find feasible solutions in all test cases. Relative to MFSPart (1.00), the normalized averages are:

Baseline [21]: MFSPart : MFSPart-Ensemble =

TABLE X: Results on Titan23 (MFS1, 8 FPGAs) — **N**, $H_{\max}$ = 2, FPGA capacity = '$2C$'.

Design	[21] Cut (K)	[21] Con (K)	[21] Hop $\times 10^{-3}$	[21] Time (s)	MFSPart Cut (K)	MFSPart Con (K)	MFSPart Hop $\times 10^{-3}$	MFSPart Time (s)	MFSPart-Ensemble Cut (K)	MFSPart-Ensemble Con (K)	MFSPart-Ensemble Hop $\times 10^{-3}$	MFSPart-Ensemble Time (s)
1	63	8.7	230	27	**56**	**5.5**	**156**	32	**53**	**3.8**	**144**	132
2	89	3.7	458	20	**89**	**1.5**	**291**	20	**88**	**0.9**	**290**	84
3	46	3.5	267	17	**43**	**1.5**	**158**	14	**42**	**0.6**	**152**	72
4	83	6.8	221	35	**80**	**3.3**	**172**	29	**78**	**2.7**	**169**	130
5	28	9.3	92	28	**27**	**7.3**	**69**	29	**23**	**5.9**	**60**	133
6	138	6.7	390	27	**123**	**2.1**	**267**	25	**118**	**1.9**	**258**	124
7	22	5.7	69	35	**14**	**1.2**	**24**	25	**14**	**0.9**	**22**	97
8	145	7.4	181	64	**142**	**4.3**	**160**	62	**142**	**3.9**	**152**	259
9	156	10.0	259	56	164	**2.9**	**183**	55	159	**2.3**	**182**	258
10	129	8.7	195	64	**122**	**6.0**	**170**	56	**114**	**5.8**	**145**	282
11	294	14.6	400	59	**288**	**4.9**	**305**	84	**279**	**2.6**	315	128
12	152	10.7	250	62	**152**	**7.4**	**203**	70	**143**	**6.0**	**203**	266
13	300	10.5	488	64	**276**	**4.0**	**270**	62	**276**	**3.7**	273	289
14	30	6.1	30	71	**27**	**1.7**	**22**	51	**25**	**1.5**	**20**	227
15	116	14.2	99	95	**109**	**4.6**	**85**	99	**98**	**4.0**	**78**	344
16	401	32.6	354	171	**316**	**11.7**	**151**	176	319	**9.8**	153	473
17	366	14.0	224	174	368	**5.2**	**143**	164	**365**	5.9	**141**	616
18	371	26.7	196	171	**340**	**9.3**	**145**	186	**336**	**7.9**	**140**	494
19	751	71.6	517	160	833	**12.0**	**364**	177	825	12.4	**360**	464
20	349	15.7	119	260	**331**	**4.0**	**108**	266	**295**	**3.9**	**96**	650
21	708	19.8	355	228	**703**	**5.7**	**217**	194	707	**2.9**	**214**	520
22	647	50.7	365	244	680	**8.1**	**230**	264	677	**6.0**	**228**	647
Norm Avg.	1.06	2.74	1.47	1.02	**1**	**1**	**1**	**1**	**0.96**	**0.79**	**0.96**	3.58

TABLE XI: Results on the benchmark from [12] (MFS2, 43 FPGAs) — **N**, $H_{\max}$ = 2, FPGA capacity = '$8C$'.

Design	[21] Cut (K)	[21] Con (K)	[21] Hop $\times 10^{-3}$	[21] Time (s)	MFSPart Cut (K)	MFSPart Con (K)	MFSPart Hop $\times 10^{-3}$	MFSPart Time (s)	MFSPart-Ensemble Cut (K)	MFSPart-Ensemble Con (K)	MFSPart-Ensemble Hop $\times 10^{-3}$	MFSPart-Ensemble Time (s)
1	135	117	310	488	**129**	**113**	**177**	352	**112**	**101**	**156**	1592
2	202	175	340	663	**182**	**157**	**187**	468	**153**	**138**	**161**	2224
3	245	214	375	854	**190**	**155**	**163**	617	**166**	**144**	182	2886
4	236	208	248	1052	**225**	**199**	**152**	710	**223**	200	156	3394
5	345	300	359	1245	**343**	**294**	**201**	921	**271**	**242**	**163**	3794
6	352	307	313	1448	**304**	**270**	**155**	1179	**291**	**263**	**157**	4857
7	430	373	346	1648	**343**	**307**	**159**	1386	**325**	**296**	162	5693
8	423	368	314	1834	**402**	**355**	**168**	1630	**371**	**328**	**151**	6144
Norm Avg.	1.12	1.12	1.91	1.32	**1**	**1**	**1**	**1**	**0.90**	**0.92**	**0.95**	4.34

$$\text{MFS1}\begin{cases}1.06:1.00:0.96 & \text{(Cut)}\\ 2.74:1.00:0.79 & \text{(Con)}\\ 1.47:1.00:0.96 & \text{(Hop)}\\ 1.02:1.00:3.58 & \text{(Time)}\end{cases}\quad \text{MFS2}\begin{cases}1.12:1.00:0.90 & \text{(Cut)}\\ 1.12:1.00:0.92 & \text{(Con)}\\ 1.91:1.00:0.95 & \text{(Hop)}\\ 1.32:1.00:4.34 & \text{(Time)}\end{cases}$$

Similarly, the conclusion mentioned above still holds true here.

Overall, our experiments show that MFSPart and MFSPart-Ensemble can flexibly handle the optional fixed-node constraint, consistently finding feasible solutions under a given $H_{\max}$ while maintaining high solution quality in terms of the *driver-sink pair cut size*, the *connectivity metric*, and the *mean hop count*. Notably, by adopting a solution ensembling technique called *cut overlay clustering* during the uncoarsening & refinement phase, MFSPart-Ensemble is able to better address complex FPGA system topologies and deliver superior results, albeit at the cost of additional running time.

C. Further Analysis

1) Parameter Sensitivity Analysis: To evaluate robustness, we conducted sensitivity analysis on parameters using the Titan23 benchmark. In Fig. 7, the x-axis shows parameter settings (the middle column indicates our experimental values), and the y-axis represents the normalized overall optimization objective. In the coarsening stage, parameters r and m address the fixed-node constraint. Setting them too low (e.g., $r = 0, m = 1$) reduces feasibility to 86%–91%, while excessively high values degrade quality. The coarsest hypergraph size parameter τ proves robust; $\tau = 30$ yields stable results with negligible impact from further reduction. In the subsequent phases, η effectively improves solutions, λ and μ show moderate impact, though very low λ reduces feasibility. For node movement, $d = 1$ significantly impairs quality, while $d = 3$ offers no benefit over $d = 2$. The *early-stopping* strategy proves highly efficient, reducing runtime significantly (ratios of $0.90 : 1 : 1.60$) while maintaining quality. Overall, the framework demonstrates strong robustness across parameter variations.

TABLE XII: Comparison of driver-sink pair cut size with [12] and [25] under their experimental settings.

Case	[12]	[25]	MFSPart	MFSPart-E*
1	149065	6469	**2413**	**1986**
2	162611	4959	**1946**	**1608**
3	139118	5612	**2170**	**1567**
4	350546	6224	**2108**	**1728**
5	177704	5480	**2093**	**1628**
6	149259	5075	**1824**	**1636**
7	304398	6355	**2748**	**1673**
8	270277	5233	**2211**	**1553**
Norm Avg.	92.10	2.60	**1**	**0.77**

* MFSPart-E denotes MFSPart-Ensemble.

2) Justification for Local Move Restriction: To validate the efficiency of restricting node moves to local neighbors, we analyzed the distribution of move distances during the partitioning process without distance constraints. As illustrated in the comparison below, the physical topologies of MFS1 and MFS2 naturally contain a significant number of long-distance FPGA pairs (with $d \geq 2$). However, the actual moves observed in our experiments are overwhelmingly local, with 93.65% of moves occurring at $d = 1$ and virtually zero moves (0.01%) at $d \geq 3$. This stark contrast between the inherent topological distribution and the algorithmic behavior confirms that restricting the search scope to local neighborhoods significantly enhances efficiency without compromising the solution quality.

$$\text{Dist. Ratio } (1:2:\geq 3)\begin{cases}11:11:6 & \text{(MFS1 Topology)}\\ 214:614:75 & \text{(MFS2 Topology)}\\ 93.65\%:6.34\%:0.01\% & \text{(Observed Moves)}\end{cases}$$

3) Comparison under the Settings of [12] and [25]: To enhance the persuasiveness of our algorithm, we compared our methods with existing works [12], [25] on the MFS2 benchmark under their experimental settings: $H_{\max} = 1$, infinite capacity, initial fixed nodes, and focus on the *driver–sink pair cut size*. As shown in Table XII, MFSPart demonstrates robust superiority: it achieves an order-of-magnitude improvement over [12], reducing the normalized average cut size to 1/92.10, and significantly outperforms [25] with a reduction to 1/2.60. The MFSPart-Ensemble strategy further refines solution quality, achieving the lowest normalized average cut size of 0.77. These results indicate that MFSPart's effectiveness stems from its advanced partitioning framework and demonstrate robustness across different experimental setups.

4) Further Evaluation on the $H_{\max} = 3$ Scenario: We further extend our evaluation to the $H_{\max} = 3$ scenario using the configurations from Setups 1–4. The results are summarized in Table XIII. As shown, our method consistently

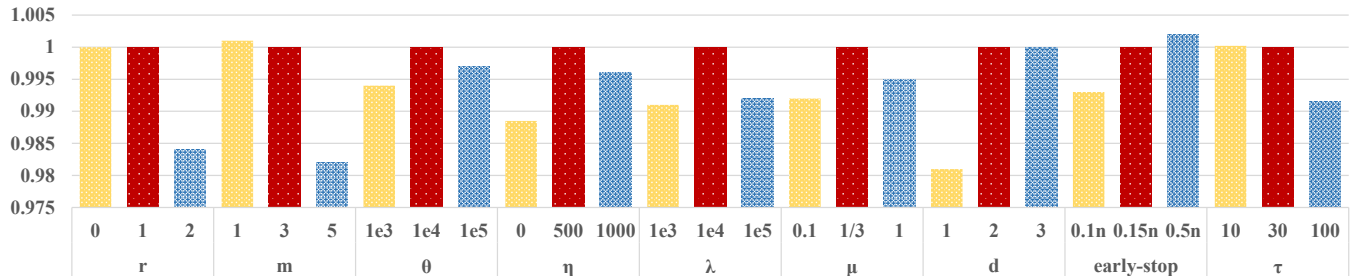


Fig. 7: Parameter sensitivity analysis: Solution quality vs. parameter settings.

TABLE XIII: Comparison on the $H_{\max} = 3$.

Setup	Baselines*				MFSPart	MFSPart-Ensemble			
	Cut	Con	Hop	Time	All metrics	Cut	Con	Hop	Time
MFS1-5C-F	1.14	3.15	1.31	1.19	**1**	**0.96**	**0.8**	**0.95**	4.24
MFS2-35C-F	1.11	1.11	1.05	1.41	**1**	**0.9**	**0.9**	**0.84**	4.56
MFS1-2C-N	1.1	2.92	1.55	1.16	**1**	**0.97**	**0.78**	**0.95**	4.83
MFS2-8C-N	1.07	1.07	1.97	1.24	**1**	**0.91**	**0.91**	**0.79**	4.35

* The Baseline column corresponds to [26] for the fixed-node case (F) and [21] for the non-fixed-node case (N).

achieves superior solution quality compared to the baselines, demonstrating its robustness across different configurations.

VI. Conclusion

We propose a generalized multi-FPGA partitioning formulation that enhances adaptability by replacing the conventional "one-hop constraint" with a user-defined threshold. This formulation supports FPGA capacity and fixed-node constraints, jointly optimizes *driver–sink pair cut size*, *connectivity*, and *mean hop count*. Based on this, we introduce MFSPart, a framework that improves runtime and quality by performing propagation solely on the coarsest graph and employing other novel strategies across all phases. To further mitigate local optima, we extend this to MFSPart-Ensemble, which leverages cut overlay clustering. It delivers superior solution quality at the cost of higher runtime. This work establishes a promising new paradigm for multi-FPGA system partitioning.

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] M. Inagi, Y. Takashima, and Y. Nakamura, "Globally optimal time-multiplexing in inter-fpga connections for accelerating multi-fpga systems," in *2009 International Conference on Field Programmable Logic and Applications*. IEEE, 2009, pp. 212–217.

[3] 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.

[4] J. Xu, C. Pei, S. Tong, and W. Yu, "Efficient hypergraph modeling of vlsi circuits for the mfs-based emulation and simulation acceleration," in *Proceedings of the 30th Asia and South Pacific Design Automation Conference*, 2025, pp. 1314–1320.

[5] 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.

[6] S. Liang, J. Lin, D. Liu, W. Lin, P. Gao, Y. Ma, T. Wu, X. Xiong, and S. Cai, "Rtmf: Routing based on tdm for multi-fpga system," *ACM Transactions on Design Automation of Electronic Systems*, 2025.

[7] 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.

[8] 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.

[9] I. Bustany, A. B. Kahng, I. Koutis, B. Pramanik, and Z. Wang, "Specpart: A supervised spectral framework for hypergraph partitioning solution improvement," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, 2022, pp. 1–9.

[10] A. O. Balkan, *Mesh-of-trees interconnection network for an explicitly multi-threaded parallel computer architecture*. University of Maryland, College Park, 2008.

[11] T. Ueno and K. Sano, "Vcsn: Virtual circuit-switching network for flexible and simple-to-operate communication in hpc fpga cluster," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 16, no. 2, pp. 1–32, 2023.

[12] 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.

[13] 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.

[14] U. Farooq, R. Chotin-Avot, M. Azeem, M. Ravoson, M. Turki, and H. Mehrez, "Inter-fpga routing environment for performance exploration of multi-fpga systems," in *Proceedings of the 27th International Symposium on Rapid System Prototyping: Shortening the Path from Specification to Prototype*, 2016, pp. 107–113.

[15] 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.

[16] M. Turki, Z. Marrakchi, H. Mehrez, and M. Abid, "Iterative routing algorithm of inter-fpga signals for multi-fpga prototyping platform," in *International Symposium on Applied Reconfigurable Computing*. Springer, 2013, pp. 210–217.

[17] W.-K. Liu, M.-H. Chen, C.-M. Chang, C.-C. Chang, and Y.-W. Chang, "Time-division multiplexing based system-level fpga routing," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–6.

[18] Z. Xie, Y.-H. Huang, G.-Q. Fang, H. Ren, S.-Y. Fang, Y. Chen, and J. Hu, "Routenet: Routability prediction for mixed-size designs using convolutional neural network," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–8.

[19] T.-W. Lin, W.-C. Tai, Y.-C. Lin, and I. H.-R. Jiang, "Routing topology and time-division multiplexing co-optimization for multi-fpga systems," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[20] D. Zheng, X. Zhang, C.-W. Pui, and E. F. Young, "Multi-fpga co-optimization: Hybrid routing and competitive-based time division multiplexing assignment," in *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, 2021, pp. 176–182.

[21] B. Li, S. Bi, H. You, Z. Qi, G. Guo, R. Sun, and Y. Zhang, "Mapart: An efficient multi-fpga system-aware hypergraph partitioning framework," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[22] W. N. Hung and R. Sun, "Challenges in large fpga-based logic emulation systems," in *Proceedings of the 2018 International Symposium on Physical Design*, 2018, pp. 26–33.

[23] H. Gao, C. Dai, J. Ji, B. Yan, Q. Zhao, J. Yuan, F. Li, L. Li, and Y. Fu, "Fpga-par: An efficient algorithm for elegant partitioning in multi-fpga systems," in *2025 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2025, pp. 1–5.

[24] 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.

[25] B. Li, H. You, S. Bi, and Y. Zhang, "An efficient hypergraph partitioner under inter-block interconnection constraints," in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2024, pp. 1–6.
[26] S. Tong, H. Li, J. Xu, C. Pei, W. Yu, S. Liu, and J. Shen, "Easypart: An effective and comprehensive hypergraph partitioner for fpga-based emulation," in *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, 2024, pp. 1–9.
[27] 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.
[28] H. Wei, O. Bhilare, H. Waqar, and J. H. Anderson, "Cad techniques for noc-connected multi-cgra systems," in *Proceedings of the 14th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies*, 2024, pp. 109–118.
[29] 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 *Proceedings of the International Conference on Computer-Aided Design*, 2018, pp. 1–7.
[30] C. Ababei, N. Selvakkumaran, K. Bazargan, and G. Karypis, "Multi-objective circuit partitioning for cutsize and path-based delay minimization," in *Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*, 2002, pp. 181–185.
[31] M.-H. Chen, Y.-W. Chang, and J.-J. Wang, "Performance-driven simultaneous partitioning and routing for multi-fpga systems," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 1129–1134.
[32] M. Deveci, K. Kaya, B. Uçar, and Ü. V. Çatalyürek, "Hypergraph partitioning for multiple communication cost metrics: Model and methods," *Journal of Parallel and Distributed Computing*, vol. 77, pp. 69–83, 2015.
[33] I. Bustany, A. B. Kahng, I. Koutis, B. Pramanik, and Z. Wang, "K-specpart: Supervised embedding algorithms and cut overlay for improved hypergraph partitioning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 4, pp. 1232–1245, 2023.
[34] 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.
[35] 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.

Yugao Zhu received the Bachelor degree in electronic engineering from Tsinghua University, Beijing, China, in 2022. He is currently pursuing the Ph.D. degree with the Department of Electronic and Computer Engineering, Hong Kong University of Science and Technology, Hong Kong. His research interests include graph algorithms and VLSI design.

Wenji Fang is currently a Ph.D. student with the Department of Electronic and Computer Engineering at the Hong Kong University of Science and Technology. He received his M.Phil. degree in Microelectronics from the Hong Kong University of Science and Technology (Guangzhou) in 2024, and his B.Eng. degree from Nanjing University of Aeronautics and Astronautics in 2021. His research interests include Electronic Design Automation (EDA) and VLSI design verification.

Yao Lu received the B.E. degree from the School of Electronic Science and Engineering, Southeast University, Nanjing, China, in 2020, and the master degree from the School of Microelectronics, Fudan University, Shanghai, China, in 2023. She is currently pursuing the Ph.D. degree with the Department of Electronic and Computer Engineering, The Hong Kong University of Science and Technology, Hong Kong. Her current research interests focus on machine learning applications in EDA.

Shang Liu received the B.E. degree in Automation Science and Electrical Engineering from Beihang University, Beijing, China, in 2023. He is currently pursuing the Ph.D. degree with the Department of Electronic and Computer Engineering, Hong Kong University of Science and Technology, Hong Kong. His research interests include agile VLSI design methodologies and Artificial Intelligence.

Yanzhen Zhu received his B.Eng. degree in Materials Forming and Control Engineering from Guangdong University of Technology in 2023. He is currently a visiting student in the Department of Electronic and Computer Engineering at The Hong Kong University of Science and Technology. His current research interests focus on floorplan and placement and the application of deep learning in EDA.

Qijun Zhang received the B.Eng. degree from Tongji University, Shanghai, China, in 2022. He is currently a Ph.D. student in the Department of Electronic and Computer Engineering (ECE) at the Hong Kong University of Science and Technology (HKUST). His research interests include Computer Architecture and Electronics Design Automation.

Zhiyao Xie is an Assistant Professor of the Department of Electronic and Computer Engineering (ECE) at the Hong Kong University of Science and Technology (HKUST). Zhiyao received his Ph.D. degree from Duke University in 2022 and B.Eng. from City University of Hong Kong in 2017. His research interests include machine learning algorithms for EDA and VLSI design. He has received multiple prestigious awards, including the IEEE/ACM MICRO 2021 Best Paper Award, ACM SIGDA SRF Best Research Poster Award 2022, ASP-DAC 2023 Best Paper Award, ACM Outstanding Dissertation Award in EDA 2023, EDAA Outstanding Dissertation Award 2023, and the 2023 Early Career Award from Hong Kong Research Grants Council (RGC).