RePart: Efficient Hypergraph Partitioning with Logic Replication Optimization for Multi-FPGA System

Zizhuo Fu[1,3,5†], Yifan Zhou[1,3,5†], Zhaoxin Lu[1,5†], Guangyu Sun[1,2,4],
Runsheng Wang[1,2,4], Meng Li[3,1,2,4*], Yibo Lin[1,2,4*]

[1]School of Integrated Circuits, *Peking University* [2]Institute of Electronic Design Automation, *Peking University*
[3]Institute for Artificial Intelligence, *Peking University* [4]Beijing Advanced Innovation Center for Integrated Circuits
[5]School of Electronics Engineering and Computer Science, *Peking University*

***Abstract*—Multi-FPGA systems (MFS) are widely adopted for VLSI emulation and rapid prototyping. In an MFS, FPGAs connect only to a limited number of neighbors through bandwidth-constrained links, so inter-FPGA communication cost depends on network topology. This setting exposes two fundamental limitations of existing MFS-aware partitioning methods: conventional hypergraph partitioners focus solely on cut size and ignore topological structure, and they leave substantial FPGA resources unused due to conservative balance margins. We present RePart, a fully customized multilevel hypergraph partitioning framework for MFS that integrates logic replication with topology-aware optimization. RePart introduces three coordinated innovations across the multilevel pipeline: FPGA-aware dynamic coarsening, heat-value guided assignment, and replication-deletion supported refinement. Extensive experiments on the Titan23 and EDA Elite Challenge Contest benchmarks show that RePart reduces total hop distance by 52.3% on average over state-of-the-art hypergraph partitioners with a 11.1× speedup, and outperforms the EDA Elite Challenge winners. Code is available at: https://github.com/Welement-zyf/RePart.**

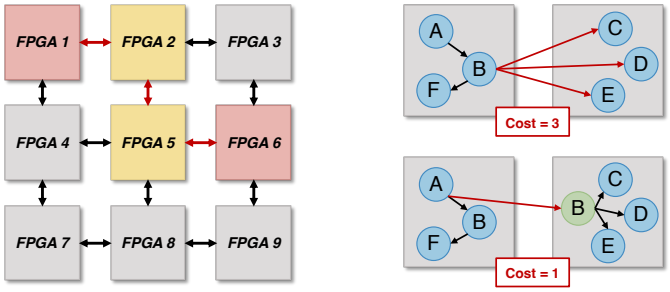
***Index Terms*—Multi-FPGA system, hypergraph partitioning, logic replication**

I. Introduction

Field-Programmable Gate Arrays (FPGAs) are widely used in logic verification, where FPGA prototyping enables high-speed, cycle-accurate validation of complex designs [1]. Recently, FPGAs have also attracted growing interest for deploying large language models, which often need to be distributed across multiple devices due to their high compute and memory demands [2]. As design complexity increases, the demand for Multi-FPGA Systems (MFS) has grown substantially [3]. In such systems, how design components are partitioned and mapped to individual FPGAs largely determines overall performance.

Design circuit netlists can be represented as hypergraphs, naturally formulating the problem as hypergraph partitioning [4]. Hypergraph partitioning is NP-hard, and has been extensively studied. Early approaches such as the KL [5] and FM [6] algorithms introduced iterative node-swapping techniques. Later advancements including hMETIS [7], PaToH [8], and KaHyPar [9] adopted multilevel frameworks comprising coarsening, initial partitioning, and refinement phases, substantially improving both efficiency and quality [10].

† These authors contributed equally to this work. * Corresponding authors: Yibo Lin (yibolin@pku.edu.cn) and Meng Li (meng.li@pku.edu.cn).



(a) Multi-FPGA System (b) Logic Replication

Fig. 1: (a) Interconnect topology of a multi-FPGA system with heterogeneous resource constraints. (b) Communication reduction through strategic logic replication.

However, hypergraph partitioning for MFS presents distinct characteristics compared to traditional approaches [7], [9]:

❶ **Topology Awareness.** Traditional partitioning algorithms focus on minimizing cut-size, which is insufficient for MFS [11]. Due to limited I/O resources, FPGAs form a specific topological network rather than a full-mesh, as shown in Fig. 1(a). For non-adjacent FPGA pairs, signals must traverse intermediate FPGAs as hops, increasing transmission delays. Additionally, Time-Division-Multiplexing (TDM) [12], [13] is adopted to multiplex signal paths over single wires. As communication increases, both TDM ratio and latency rise. Thus, this work focuses on minimizing total hop distance [11], [14] to optimize performance.

❷ **MFS-specific Constraints.** FPGAs contain various hardware resources [15], including Flip-Flops (FFs), Look-Up Tables (LUTs), and Digital Signal Processors (DSPs), each with distinct capacity limits. Circuit nodes must be assigned without exceeding any resource limit on each FPGA. Furthermore, inter-FPGA communication relies on I/O pins [15], imposing additional constraints. Many applications also constrain the maximum hop count to meet timing requirements [14].

Recent works including TopoPart [16], MaPart [14], and EasyPart [17] have extended traditional approaches by incorporating MFS constraints. These methods typically model FPGA resource constraints through an imbalance factor ϵ, leaving FPGA resources underutilized after partitioning. We address this limitation by introducing logic replication [18] to

maximize hardware utilization. As shown in Fig. 1(b), replication reduces inter-FPGA communication by enabling local access to replicated nodes. However, integrating logic replication into multilevel partitioning requires careful balancing, as replicated nodes consume resources that may compromise subsequent refinement operations.

This paper presents RePart, a fully customized multilevel hypergraph partitioning framework for MFS that integrates logic replication with topology-aware optimization. Our contributions are summarized as follows:

1) We adapt logic replication for multilevel hypergraph partitioning in MFS environments with a fully customized implementation that natively supports MFS constraints.
2) We propose three key innovations in the multilevel partitioning framework: (1) dynamic coarsening with FPGA-aware scoring, (2) heat-value guided assignment, and (3) logic replication and deletion supported refinement, collectively improving partitioning quality.
3) We conducted extensive experiments to validate our approach, which demonstrates 52.3% average reduction in total hop distance compared to state-of-the-art partitioners with 11.1× speedup. Our solution also outperforms winners of the EDA Elite Challenge Contest [19].

II. Preliminary

A. Multi-FPGA System and Modeling

Our work targets hypergraph partitioning with logic replication under MFS constraints. The problem input comprises the FPGA network and the Design Under Test (DUT).

The FPGA network specifies the number of FPGAs and their bidirectional connectivity. We define hop distance as the shortest path length between two FPGAs in the network topology. The DUT consists of circuit components and interconnecting nets, where signals propagate from source nodes to all drain nodes. We model the DUT connectivity as a hypergraph, where the partitioning solution directly determines the MFS deployment strategy. Table I summarizes key notations used throughout this work.

Our optimization objective minimizes inter-FPGA communication cost, quantified through total hop distance [19]:

$$\text{hop\_distance}(e) = \sum_{\hat{v} \in N(e)} \text{hop}(\text{part}(\text{source}(e)), \hat{v})$$

$$\text{total\_hop\_distance} = \sum_{e \in E} w_e \cdot \text{hop\_distance}(e)$$

This metric directly correlates with communication latency, bandwidth contention, and TDM ratio in MFS [11], [20]. Logic replication, illustrated in Fig. 1(b), provides a mechanism to reduce communication cost by replicating source nodes across multiple FPGAs. While increasing resource consumption, strategic replication eliminates inter-FPGA signal routing, as shown where replicating node B onto FPGA 2 removes three cut edges at the cost of one additional communication link to node A.

Table I: Terminology and Notation.

Term	Definition
$H(V, E)$	Hypergraph with vertex set V and hyperedge set E
v, e	Circuit unit (vertex) and circuit net (hyperedge)
$\|e\|$	Number of vertices in hyperedge e
$\omega_i(v)$	Type-i resource usage of vertex v
$\overline{C_i}$	Average FPGA capacity for resource type i
w_e	Weight of net e (number of signals in the net)
source(e), drain(e)	Source vertices and drain vertices set of e
$I(v)$	Set of hyperedges containing vertex v
$N(e)$	Set of FPGAs containing drain vertices of e
part(v)	FPGA assignment of vertex v
$G(\hat{V}, \hat{E})$	MFS network with FPGA set $\hat{V}$ and connection set $\hat{E}$
$\hat{v}$	An individual FPGA in $\hat{V}$
hop($\hat{u}, \hat{v}$)	Hop distance between FPGAs $\hat{u}$ and $\hat{v}$
α, ϵ, ℓ	Penalty exponent, imbalance factor, coarsening level

B. Multilevel Hypergraph Partitioning Framework

We adopt the multilevel partitioning paradigm used by state-of-the-art tools such as KaHyPar [21] and hMETIS [7]. This framework operates through three sequential phases:

Coarsening: The hypergraph undergoes iterative reduction through vertex aggregation, creating a hierarchy of progressively smaller hypergraphs. Strongly connected vertices merge into hypernodes, reducing problem complexity while preserving essential structural properties.

Assignment: A coarse-level hypergraph receives an initial partition using specialized algorithms. Unlike traditional partitioning into equivalent clusters, our MFS-specific task assigns hypernodes to heterogeneous FPGAs with distinct resource capacities and topological positions.

Refinement: The partition propagates back through the hierarchy to the original hypergraph. Local optimization techniques, including KL [5] and FM [6] algorithms, progressively improve solution quality while respecting resource and communication constraints at each uncoarsening level.

III. Methodology

A. Overview

RePart adopts a multilevel hypergraph partitioning framework comprising three phases: coarsening, assignment, and refinement. We develop a fully customized implementation tailored for multi-FPGA systems, addressing the limitations of conventional partitioners through three key innovations:

Dynamic Coarsening with FPGA-Aware Scoring. Traditional coarsening ignores multi-dimensional FPGA constraints, often producing infeasible solutions. We introduce a dynamic merging strategy that balances connectivity preservation with resource utilization, adapting penalty weights across coarsening levels to enable effective initial partition generation.

Heat-Value Guided Assignment. Unlike homogeneous partitioning, MFS assignment must consider heterogeneous resource constraints and network topology. We propose a heat-value metric that quantifies architectural significance of both FPGAs and nodes, coupled with a deep backtracking mechanism to explore diverse solution subspaces.

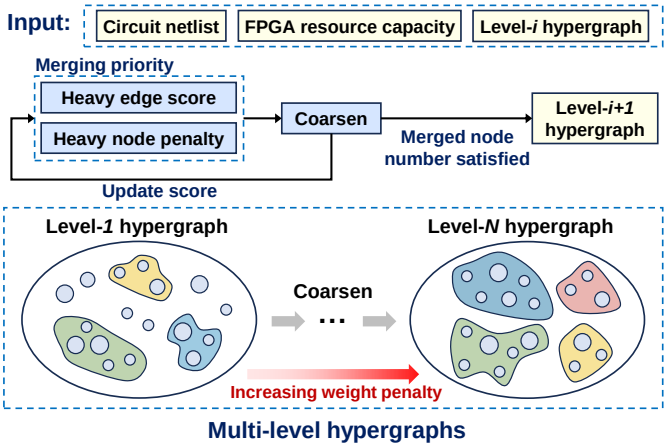


Fig. 2: Dynamic Coarsening Process.

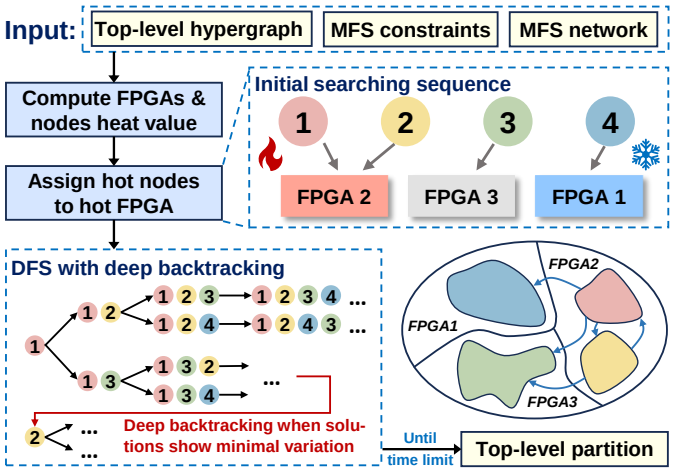


Fig. 3: Heat-Value Guided Assignment Process.

Replication and Deletion Supported Refinement. Beyond traditional move and exchange operations, we introduce replication to exploit spare FPGA resources and deletion to reclaim space consumed by coarse-level replications. These operations are coordinated through a $1 + 3K$-Heap structure with incremental gain updates for efficient optimization.

B. Dynamic Coarsening with FPGA-Aware Scoring

Coarsening merges strongly-connected nodes into hypernodes across multiple hierarchy levels, enabling later refinement at varying granularities. Our approach balances two objectives: preserving connectivity within clusters while maintaining balanced resource utilization for effective assignment.

1) Connectivity and Balance Scoring: We evaluate merging candidates using connectivity strength and resource balance. For node pair (u, v), the **HeavyEdgeScore** [7], [8], [21] measures connectivity:

$$r(u,v) = \sum_{e \in I(v) \cap I(u)} \frac{w_e}{|e| - 1}$$

where $I(v)$ denotes hyperedges containing v, w_e is edge weight, and $|e|$ is edge size. This metric prioritizes high-weight, low-degree connections.

The **HeavyNodePenalty** enforces multi-dimensional resource balance:

$$p(u,v) = \left(\sum_{i=1}^{k} \frac{\omega_i(u) \cdot \omega_i(v)}{\overline{C}_i^2} \right)^{\alpha}$$

where $\omega_i(v)$ represents type-i resource usage of node v, $\overline{C}_i$ is the average FPGA capacity for resource type i, k is the number of resource types, and α is the penalty exponent.

2) Adaptive Penalty Adjustment: Early coarsening should emphasize connectivity, while later stages require balanced clusters. We dynamically adjust α across levels ℓ:

$$\alpha = \alpha_0 + \frac{\Delta_\alpha \cdot \ln 2}{\ln\left(\frac{N_{\text{init}}+1}{N_{\text{final}}}\right)} \cdot \ell$$

where α_0 and Δ_α define baseline and increment parameters, N_{init} and N_{final} specify initial and final node counts. This logarithmic scaling smoothly transitions from connectivity-focused to balance-focused merging.

The final merging priority combines both criteria:

$$\Psi(u,v) = \frac{r(u,v)}{p(u,v)}$$

As illustrated in Fig. 2, early levels create tightly-connected but uneven clusters (left), while later levels enforce balanced resource distribution (right). Our experiments show this adaptive framework improves initial solution quality by 18.7% over static penalty methods, as shown in Sec. IV-C.

C. Heat-Value Guided Assignment

Traditional partitioning treats all targets as homogeneous clusters, ignoring architectural heterogeneity. In MFS, each FPGA has distinct resource constraints and topological positions. We reformulate this as an architecture-aware assignment problem using heat-value metrics and deep backtracking.

1) Heat-Value Metrics: We quantify architectural significance of both FPGAs and nodes:

FPGA Heat Value evaluates topological centrality:

$$\text{Hop\_sum}(\hat{v}) = \sum_{\hat{u} \in \hat{V} \setminus \{\hat{v}\}} \begin{cases} \text{hop}(\hat{v}, \hat{u}) & \text{if hop} \le \text{Hop}_{\max} \\ \beta \cdot \text{Hop}_{\max} & \text{otherwise} \end{cases}$$

$$\text{Heat}_{\text{FPGA}}(\hat{v}) = \frac{\sum_{i=1}^{k} \omega_i(\hat{v})^2}{\text{Hop\_sum}(\hat{v})}$$

where $\hat{v}$ denotes an FPGA, $\hat{V}$ is the FPGA set, $\text{hop}(\hat{v}, \hat{u})$ is hop distance between FPGAs, $\text{Hop}_{\max}$ is the maximum hop constraint, and β is the penalty coefficient (set to 2).

Node Heat Value measures connectivity and resource criticality:

$$\text{Heat}_{\text{node}}(v) = \sum_{e \in I(v)} w_e \cdot \sum_{i=1}^{k} \omega_i(v)^2$$

High heat-value nodes are prioritized for placement on central FPGAs to minimize communication cost (Fig. 3).

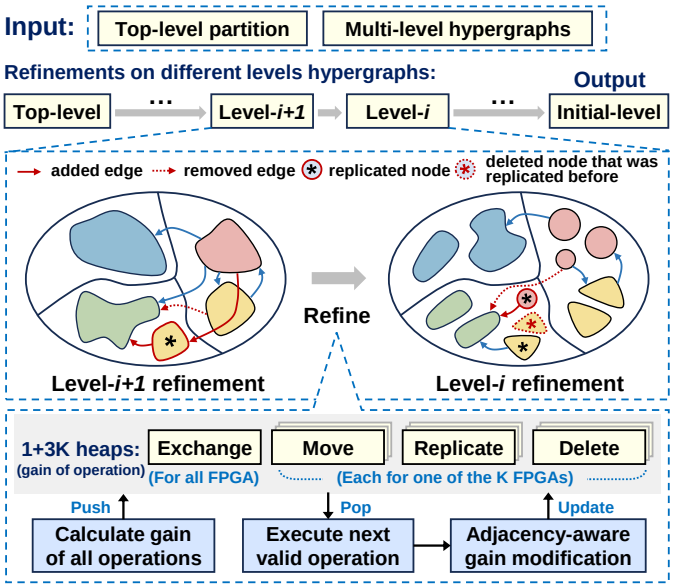


Fig. 4: Logic Replication Supported Refinement Process.

Table II: MFS configurations.

Testcase	case01	case02	case03	case04	case05	case06
#FPGAs	4	8	32	64	4	8
#Links	3	11	88	173	3	28
#Types	8	8	8	8	8	8
Testcase	case07	case08	case09	case10	SampleInput	synopsys02
#FPGAs	32	32	64	64	8	56
#Links	92	92	177	177	11	157
#Types	8	8	8	8	1	1

Table III: Total hop distance comparison on the EDA Contest benchmark [19].

Testcase	#Nodes	#Edges	KaHyPar	2nd place	1st place	RePart
case01	16	13	17	11	8	**8**
case02	600	1239	3608	2321	2456	**2394**
case03	11451	31071	13909	9870	7371	**6199**
case04	1000000	953817	57850	46194	19469	**14077**
case05	1600	2157	299	182	147	**118**
case06	1053	2447	409	347	300	**300**
case07	40000	45504	7843	7654	**5547**	5702
case08	25000	27482	6370	5898	4324	**3631**
case09	240000	253232	46335	45720	33276	**31090**
case10	900000	1108491	24241	27983	14859	**12027**
Avg.ratio	–	–	1.00	0.91	0.545	0.47

2) Deep Backtracking Exploration: Assignment performs depth-first search with three pruning conditions: (1) exceeding current best total hop distance, (2) violating communication utilization, and (3) exceeding FPGA resource capacity.

Solution spaces contain multiple local minima regions. Conventional methods often start from an arbitrary initial partitioning, which confines the subsequent refinement to a single local optimum. Our key insight is that assignment should explore diverse solution subspaces rather than merely finding low-cost solutions. Solutions with better structural properties enable superior refinement outcomes.

As shown in Fig. 3, we implement deep backtracking that triggers when the THD of consecutive solutions show minimal variation ($< 2\%$ normalized cost difference). This mechanism performs aggressive pruning to escape local valleys, enabling cross-valley exploration. Heat-value scoring provides theoretical grounding: early assignment variations of high heat-value nodes fundamentally reshape solution trajectories, creating distinct subspaces with unique refinement potential.

D. Logic Replication and Deletion Supported Refinement

Refinement optimizes the initial partition through multi-level adaptation, leveraging hierarchical structural properties to progressively improve solution quality.

1) Four-Operation Framework: Beyond traditional **move** and **exchange** operations [21], we introduce two MFS-specific operations. The **replicate** operation replicates nodes across multiple FPGAs, utilizing remaining resources to reduce inter-FPGA communication by enabling local signal access. However, replicated nodes consume FPGA resources, constraining fine-grained refinement space.

The **delete** operation addresses this by removing unwanted replicated nodes to reclaim resources for other operations. Delete executes when its gain is zero or positive, prioritizing resource recovery over immediate gain to create optimization headroom for subsequent steps.

2) Multi-Operator Coordination via $1 + 3K$-Heap: We coordinate four operation types through a $1 + 3K$-Heap structure, as illustrated in Fig. 4. One heap tracks cross-FPGA node exchange opportunities. For move, replicate, and delete operations, we maintain K heaps for each operation type, where K is the number of FPGAs. Each heap stores operation gains when targeting a specific FPGA as the destination.

The algorithm iteratively applies the highest-gain operation from these heaps. Each operation execution triggers online updates to total hop distance, resource usage, and communication utilization. Operation gains are dynamically updated through adjacency-aware propagation, where node modifications trigger localized gain recalculations for neighboring hyperedges.

This incremental update mechanism, though algorithmically complex in multi-FPGA contexts, achieves significant speedup. Our implementation delivers an average $2.27\times$ acceleration compared to naive full recalculation while maintaining identical solution quality, making refinement computationally efficient for large-scale problems.

IV. Experimental Results

We implemented RePart in C++ and compiled it with g++ 11.4.0 on a machine equipped with an Intel Xeon PLATINUM 8558P 2.7-GHz CPU running Ubuntu 22.04.

A. Experimental Setup

Hypergraphs. Experiments are conducted on two benchmarks. The first is Titan23 [22], a widely used suite of circuit netlists in which all hyperedges are unweighted and treated as equivalent. The second comprises 10 test cases from Problem 3 of the Integrated Circuit EDA Elite Challenge Contest [19], where each hyperedge carries a weight. The node and edge counts for each benchmark are listed in the result tables.

MFS Instances. For Titan23, we use two MFS instances from the ICCAD'19 Contest [23]: *SampleInput* and *synopsys02*. For the EDA Contest test cases, each case comes with its own unique FPGA system. These systems are substantially more complex: FPGAs are heterogeneous, each subject to eight distinct resource constraints (FF, LUT, BUFG, TBUF, DCM, BRAM, DSP, and PP), and each FPGA has an individual limit on the number of inter-FPGA connections. The MFS configurations are summarized in Table II, where #FPGAs denotes the number of FPGAs, #Links denotes the number of inter-FPGA connections, and #Types denotes the number of distinct resource constraint categories per FPGA.

Constraints. All 10 EDA Contest cases impose strict multi-resource constraints on every FPGA. For Titan23, the imbalance factor ϵ is set to 0.2.

B. Results Analysis

1) RePart-mini and RePart: RePart-mini follows the same multilevel pipeline as KaHyPar but is more concise and intuitive. It omits the topology-aware coarsening and heat-value guided assignment strategies proposed in this work, and only retains node replication in the refinement phase to isolate the effect of logic replication. RePart incorporates all proposed optimizations and achieves significant improvements in both total hop distance and runtime over KaHyPar.

2) Total Hop Distance Reduction: Table III reports results on the EDA Contest benchmark, where total hop distance is the primary metric. RePart-mini, equipped solely with node replication, achieves an average total hop distance of 12540.4, which is 77.9% of KaHyPar's 16088.1, confirming that replication alone yields a meaningful gain. RePart further reduces the average to 7554.6, only 47.0% of KaHyPar's result. Compared to the contest winners, RePart achieves a 14.0% reduction over the 1st-place solution and a 48.3% reduction over the 2nd-place solution.

Table IV presents results on the Titan23 benchmark under the *synopsys02* MFS with $\epsilon = 0.2$. Although Titan23 is closer to a general partitioning problem due to its unweighted edges and single resource type, our methods still deliver strong results. RePart-mini achieves a notable reduction in total hop distance over KaHyPar with significantly lower runtime. RePart reduces total hop distance by 52.3% and runtime by 98.1% compared to KaHyPar, demonstrating that the proposed optimizations are effective across both simple and complex MFS settings.

3) Cut Size Comparison: Prior works TopoPart [16], Li et al. [24], and RepPart [25] use cut size as their primary evaluation metric. However, cut size does not reflect the actual topological communication cost in an MFS when the maximum hop count exceeds one, since it treats all inter-FPGA edges equally regardless of the physical distance between FPGAs. Total hop distance captures this topology-dependent cost and is therefore a more faithful metric for real MFS deployments [19]. Because these prior methods do not support weighted hyperedges and are not open-source, we compare against them only on the Titan23 *SampleInput* instance using each paper's originally reported figures.

As shown in Table V, RePart achieves a smaller cut size than all baselines even though cut size is not its optimization objective. This result indicates that optimizing for total hop distance simultaneously yields high-quality partitions from the perspective of traditional cut-based metrics.

C. Ablation Study

We conduct an ablation study on the 10 EDA Contest test cases to evaluate the individual contribution of each proposed technique. Fig. 5 reports the normalized total hop distance

Table IV: Comparison on the Titan23 benchmark under the *synopsys02* MFS. THD denotes total hop distance; t denotes runtime in seconds.

Testcase	#Nodes	#Edges	KaHyPar		RePart-mini		RePart	
			THD	t	THD	t	THD	t
sparcT1_core	91976	92827	57806	3419	32610	**66**	**22391**	81
neuron	92290	125305	10838	2933	8546	**48**	**6239**	85
stereo_vision	94050	12708	10514	**25**	8835	50	**6655**	89
des90	111221	139557	33170	3113	18175	**56**	**15457**	66
SLAM_spheric	113115	142408	54909	633	32322	**53**	**24967**	157
cholesky_mc	113250	144948	21987	39	17818	**27**	**16356**	67
segmentation	138295	179051	28937	241	14546	**59**	**11563**	84
bitonic_mesh	192064	235328	29297	2394	18746	**77**	**12636**	95
dart	202354	223301	24432	2813	19237	**61**	**14692**	202
openCV	217453	284108	49959	2582	20874	**88**	**13376**	220
stap_qrd	240240	290123	31803	312	23558	**95**	**21289**	143
minres	261359	320540	18326	318	16427	**84**	**11347**	148
cholesky_bdti	266422	342688	38213	278	27886	**98**	**25046**	163
denoise	275638	356848	18765	801	15269	**69**	**11049**	79
sparcT2_core	300109	302663	105303	608	46209	**156**	**43738**	405
gsm_switch	493260	507821	151638	7721	69345	**259**	**46650**	469
mes_noc	547544	577664	33654	2606	21982	**224**	**16817**	379
LU_Network	635456	726999	**44283**	8206	50429	**284**	52066	607
LU230	574372	669477	81471	13659	53387	**260**	**41990**	727
sparcT1_chip2	820886	821274	98869	14491	45129	**442**	**36679**	853
directrf	931275	1374742	67096	1770	33196	**355**	**24895**	509
bitcoin_miner	1089284	1448151	89160	5149	66531	**451**	**48561**	1060
Avg.ratio			1	1	0.600	0.045	0.477	0.090

Table V: Cut size comparison on the Titan23 benchmark under the *SampleInput* MFS.

Testcase	#Nodes	#Nets	TopoPart	Li. [24]	RepPart	RePart
sparcT1_core	91976	92827	54841	1048	1185	4255
neuron	92290	125305	8815	4384	1197	**711**
stereo_vision	94050	12708	32068	388	1110	723
des90	111221	139557	87211	656	745	1221
SLAM_spheric	113115	142408	54402	7642	6987	**6691**
cholesky_mc	113250	144948	37658	428	1004	1686
segmentation	138295	179051	34092	982	1509	**940**
bitonic_mesh	192064	235328	184894	1169	1078	1893
dart	202354	223301	168197	2776	2753	**2167**
openCV	217453	284108	120428	525	1047	766
stap_qrd	240240	290123	134256	6049	3193	**1648**
minres	261359	320540	96178	13617	4512	**732**
cholesky_bdti	266422	342688	75479	578	572	1645
denoise	275638	356848	93464	595	859	1023
sparcT2_core	300109	302663	98128	4920	1023	4500
gsm_switch	493260	507821	267424	14999	10191	**7390**
mes_noc	547544	577664	297140	324	311	3405
LU230	574372	669477	134877	269	320	4878
LU_Network	635456	726999	152508	32467	9915	**5100**
sparcT1_chip2	820886	821274	207814	1417	958	2283
directrf	931275	1374742	113493	435	936	2249
bitcoin_miner	1089284	1448151	413566	43935	9345	**1125**
Avg.ratio	–	–	1	0.049	0.022	0.019

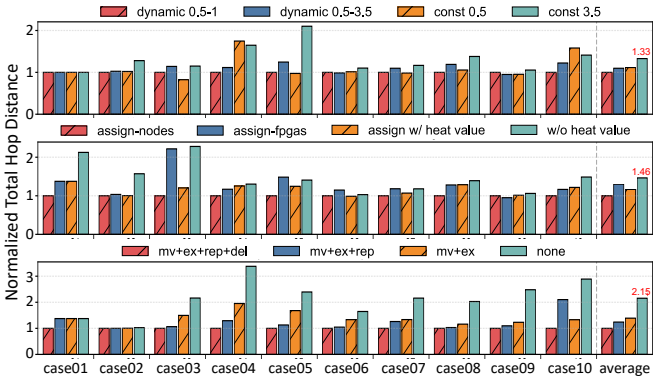


Fig. 5: Ablation study on the EDA Contest benchmark. Normalized total hop distance is shown for coarsening (top), assignment (middle), and refinement (bottom).

across three subfigures, each corresponding to one phase of the multilevel pipeline. The red bars represent the configurations that incorporate the proposed innovation for that phase.

1) Coarsening: The first subfigure in Fig. 5 examines the effect of the adaptive penalty exponent α in the FPGA-aware coarsening score. Using a fixed value of $\alpha = 3.5$ yields an average total hop distance 20% higher than the dynamic schedule that gradually increases α from 0.5 to 3.5, confirming that the adaptive range is important for guiding early-stage coarsening decisions.

2) Assignment: The second subfigure compares three variants of the heat-value guided assignment. *assign-nodes* uses four threads, each emphasizing differences among nodes during heat score evaluation. *assign-fpgas* also uses four threads but focuses instead on differences among FPGAs. *assign* is the single-threaded baseline. Since node-level variability is typically more pronounced in practice, *assign-nodes* consistently achieves the best results and is adopted in RePart.

3) Refinement: The third subfigure evaluates the contribution of each refinement operation: move (*mv*), exchange (*ex*), replication (*rep*), and deletion (*del*). Applying all four operations achieves a $2.15\times$ reduction in average total hop distance compared to no refinement, and a $1.39\times$ reduction compared to the *mv+ex*-only baseline. Case 10 highlights the necessity of the deletion operation: *mv+ex+rep* alone performs worse than *mv+ex* in this case because unconstrained replication occupies FPGA resources and limits the subsequent move and exchange operations. The deletion operation releases these resources, enabling further optimization and recovering the performance loss.

V. Conclusion

This paper presents RePart, an open-source hypergraph partitioning framework that integrates logic replication with topology-aware optimization across all phases of the partitioning pipeline. RePart effectively reduces inter-FPGA communication while exploiting available hardware resources. Experimental results demonstrate that RePart reduces total hop distance by 52.3% on average compared to state-of-the-art hypergraph partitioning algorithms, while requiring only 9.0% of its runtime. RePart also outperforms the winners of the EDA Elite Challenge Contest, confirming its effectiveness on both standard and highly constrained MFS benchmarks.

Acknowledgments

We sincerely thank Jiarui Wang, Zizheng Guo, and Benzheng Li for their valuable comments and constructive guidance in improving this manuscript. We also gratefully acknowledge the Organizing Committee of the China Postgraduate IC Innovation Competition · EDA Elite Challenge Contest and S2C for providing the datasets for this work.

References

[1] J. Ray and J. C. Hoe, "High-level modeling and fpga prototyping of microprocessors," in *Symposium on Field Programmable Gate Arrays*, 2003.

[2] S. Hong, S. Moon, J. Kim, S. Lee, M. Kim, D. Lee, and J.-Y. Kim, "Dfx: A low-latency multi-fpga appliance for accelerating transformer-based text generation," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022.

[3] R. Tessier, "Chapter 30 - multi-fpga systems: Logic emulation," in *Reconfigurable Computing*, ser. Systems on Silicon, S. Hauck and A. Dehon, Eds. Burlington: Morgan Kaufmann, 2008, pp. 637–669.

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

[5] B. W. Kernighan and S.-D. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Syst. Tech. J.*, vol. 49, pp. 291–307, 1970.

[6] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," *19th Design Automation Conference*, pp. 175–181, 1982.

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

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

[9] Y. Akhremtsev, T. Heuer, P. Sanders, and S. Schlag, "Engineering a direct k-way hypergraph partitioning algorithm," in *Workshop on Algorithm Engineering and Experimentation*, 2017.

[10] G. Karypis and V. Kumar, "Multilevel k-way hypergraph partitioning," in *Proceedings of the 36th annual ACM/IEEE design automation conference*, 1999, pp. 343–348.

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

[12] J. Babb, R. Tessier, M. Dahl, S. Hanono, D. M. Hoki, and A. Agarwal, "Logic emulation with virtual wires," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 16, pp. 609–626, 1997.

[13] P. Zou, Z. Lin, X. Shi, Y. Wu, J. Chen, J. Yu, and Y.-W. Chang, "Time-division multiplexing based system-level fpga routing for logic verification," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[14] 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*, vol. 43, pp. 3212–3225, 2024.

[15] I. Kuon, R. Tessier, and J. Rose, "Fpga architecture: Survey and challenges," *Found. Trends Electron. Des. Autom.*, vol. 2, pp. 135–253, 2008.

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

[17] 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 *International Conference on Computer Aided Design*, 2024.

[18] G. Beraudo and J. Lillis, "Timing optimization of fpga placements by logic replication," in *Proceedings 2003. Design Automation Conference (IEEE Cat. No.03CH37451)*, 2003, pp. 196–201.

[19] (2024) 2024 integrated circuit eda elite challenge contest: Contest problem 3 — hypergraph partitioning algorithm design with logic replication. [Online]. Available: https://eda.icisc.cn/en/

[20] U. Farooq, H. Mehrez, and M. K. Bhatti, "Inter-fpga interconnect topologies exploration for multi-fpga systems," *Design Automation for Embedded Systems*, vol. 22, pp. 117 – 140, 2018.

[21] 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, 2021.

[22] 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 Trans. Reconfigurable Technol. Syst.*, vol. 8, pp. 10:1–10:18, 2015.

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

[24] B. Li, H. You, S. Bi, and Y. Zhang, "An efficient hypergraph partitioner under inter - block interconnection constraints," *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1–6, 2024.

[25] H. Wu, S. Bi, J. Tang, H. Wang, and H. You, "Reppart: An efficient partitioning framework with replication technique for mfs," *2025 International Symposium of Electronics Design Automation (ISEDA)*, pp. 393–399, 2025.