FPGA-Par: An Efficient Algorithm for Elegant Partitioning in Multi-FPGA Systems

Hengyue Gao[1], Chenyang Dai[1], Jinlun Ji[2], Bin Yan[2],
Qiyue Zhao[1], Jiangtao Yuan[1], Feng Li[1], Li Li[2], Yuxiang Fu[1]
[1] School of Integrated Circuits, Nanjing University, China
[2] School of Electronic Science and Engineering, Nanjing University, China
Email: yuxiangfu@nju.edu.cn

***Abstract*—This work introduces FPGA-Par, an efficient graph partitioning algorithm for multi-FPGA systems. FPGA-Par utilizes an iterative balanced partitioning and supernodes transferring algorithm that transforms imbalanced partitioning problems into balanced ones, addressing the inefficiencies associated with traditional single-node movement methods, as well as eliminating the need for brute-force exploration of imbalance factors. This approach achieves partitions with flexible size ratios that satisfy resource constraints, resulting in improved partition quality. Experimental results demonstrate that FPGA-Par reduces time overhead by 55% compared to state-of-the-art imbalanced partitioning algorithms. Furthermore, it improves partition quality metrics, with Cut and Cut_{max} decreasing by 38% and 25%, respectively, and achieves a 1.25x increase in system frequency after mapping and routing on a multi-FPGA system.**

***Index Terms*—Graph Partitioning, Multi-FPGA Systems, Circuit Simulation**

I. Introduction

Circuit partitioning is crucial for the prototype validation of multi-FPGA systems, as its quality directly influences the system frequency [1]. Traditional graph partitioning methods [2]–[7] mostly focus on balanced partitioning. However, in a multi-FPGA system, the goal is to maximize system frequency within FPGA capacity constraints, prioritizing fewer interconnections to minimize communication overhead and boost performance.

Existing effective approaches for imbalanced partitioning problems primarily involve providing an imbalance factor, or specifying the size ratios of partitions. For instance, methods suitable for heterogeneous computing [8], as well as several widely-used balanced partitioning algorithms [7].

Some studies create imbalanced partitions using single-node movements [9]. While these algorithms avoid the exhaustive exploration of imbalance factors, they become increasingly time-consuming as the size of the graph grows. Moreover, greedy approaches are prone to getting trapped in local optima.

Currently, there is research exploring AI-based methods [10], as well as hybrid approaches [11] to address this challenge. However, these approaches are often constrained to relatively small-scale instances. Some studies [12] have also investigated solving this problem using specific constraints, yet a universally applicable method remains elusive.

The corresponding author is Yuxiang Fu (yuxiangfu@nju.edu.cn).

This work was supported in part by the National Key Research and Development Program of China (No. 2023YFB2806802) and in part by the National Nature Science Foundation of China under Grant No. 62104098 and in part by the Joint Funds of the National Nature Science Foundation of China under Grant No. U21B2032 and in part by the Natural Science Foundation of Jiangsu Province for Youth under Grant No. BK20210178.

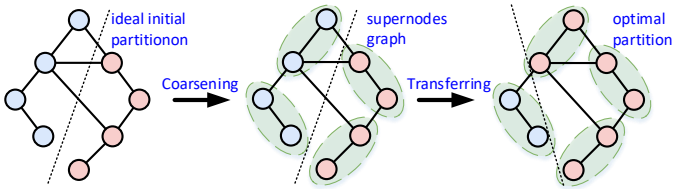

Fig. 1: The Inspiration Behind Our Method: The Generation and Scheduling of Supernodes.

In this work, we introduce FPGA-Par, an efficient and elegant graph partitioning algorithm for multi-FPGA systems. Fig. 1 provides a simple example of our inspiration, we first perform a balanced partition. Then we apply another round of balanced partitioning to create supernodes which are then strategically relocated based on constraints and gains to enhance the quality of the partition. In the specific implementation of FPGA-Par, the graph is coarsened multiple times at varying scales to facilitate flexible partition size ratios. The source code is released on Github [13].

Our main contributions in this work include:

- An adaptive graph partitioning algorithm tailored for multi-FPGA systems, FPGA-Par, is presented. It efficiently explores optimal partitions satisfying constraints.
- The transformation of imbalanced partitioning problems into iterative balanced ones, simplifying the problem elegantly.
- Experimental evaluations confirm that the proposed algorithm surpasses state-of-the-art methods in partition quality and significantly reduces computational time.

II. Preliminary

A. Multi-FPGA Systems

A multi-FPGA system has multiple FPGAs connected in various ways like mesh, star, or more complex shapes [14]. Due to delays in these connections, it's important to reduce the number of signal connections between parts of the circuit during partitioning. This helps lower communication costs

and boosts the system's speed. The system frequency can be roughly estimated using a specific formula (1) [15], [16]:

$$sys\_freq \propto \frac{100}{\frac{max\_cut}{IO} + max\_hop}. \tag{1}$$

In (1), *max_cut* represents the largest signal connections size among all interconnected pairs of FPGAs. *IO* represents the number of physical interconnect wires between FPGAs. *max_hop* represents the maximum value among all hop counts for signals that do not have direct physical connections, *IO* is a fixed value, by reducing *max_cut* we achieve a higher *sys_freq*.

Another characteristic of FPGAs is that they possess various resources. During the partitioning process, it is essential to ensure that the obtained circuit partitions can be accommodated by FPGA chips in all resources.

B. Problem Formulation

The circuit partitioning problem under a multi-FPGA system can be formulated as follows:

Given the following parameters:

1) An undirected graph $G = (V, E, w_v, w_e)$, where V is the set of vertices, each associated with a K-dimensional vector denoting its weight, $w_v : V \rightarrow \mathbb{R}^K$. The edges E, subsets of vertex pairs, are assigned weights through the function $w_e : V \rightarrow \mathbb{R}$. This graph G represents the signal interconnections and resource demands of the circuit to be partitioned, and K represents the number of resource categories in FPGA configurations.

2) A positive integer $C \in \mathbb{Z}^+$ represents the number of available FPGA chips, indicating the number of partitions for graph G.

3) A K-dimensional vector $R \in \mathbb{R}^K_{\geq 0}$ represents the resource capacities of each category in an FPGA chip.

The graph partitioning problem under multi-FPGA system involves partitioning the vertex set V into C disjoint subsets, denoted as $V_0, V_1, \ldots, V_{C-1}$, while satisfying the constraints illustrated in (2):

$$\begin{aligned} \sum_{v \in V_i} (w_v(v))_j \quad &\leq \quad R_j, \\ \forall i \in \{0, 1, 2, ..., C-1\}, \quad &\forall j \in \{0, 1, 2, ..., K-1\}. \end{aligned} \tag{2}$$

The quality of the graph partitioning is measured by the metrics Cut and Cut_{max}, which are calculated using (3) and (4), respectively:

$$Cut = \sum_{\{u,v\} \in E} w_e(u, v) - \sum_{i=0}^{C-1} \sum_{\{u,v\} \in E(V_i)} w_e(u, v). \tag{3}$$

$$\begin{aligned} Cut_{max} = \max_{i,j} \Big(\sum_{u \in V_i, v \in V_j} w_e(u, v) \quad \Big), \\ \forall i, j \in \{0, 1, 2, ..., C-1\}. \end{aligned} \tag{4}$$

Here, Cut represents the total weight of edges being cut between all subsets, indicating the overall communication overhead among subcircuits. Conversely, Cut_{max} represents the maximum total weight of edges being cut between any two subsets, highlighting the bottleneck in communication costs. To optimize the system frequency calculated by (1), joint optimization of both Cut and Cut_{max} is crucial.

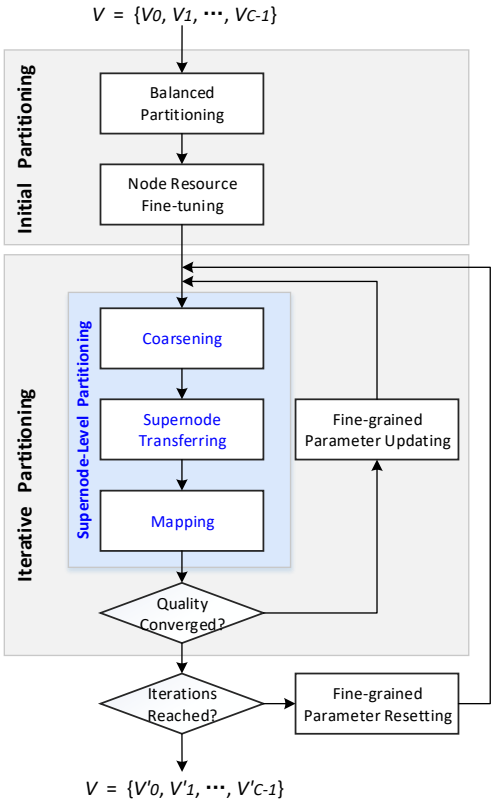

Fig. 2: Overall flow of the proposed algorithm FPGA-Par.

III. Methodology

This study introduces FPGA-Par, a novel adaptive graph partitioning algorithm for multi-FPGA systems. The overall process is depicted in Fig. 2.

A. Initial Partitioning

At the beginning of our algorithm, a balanced partitioning algorithm is utilized to generate an original partition $V = \{V_0, V_1, \ldots, V_{C-1}\}$ for the node set of the input graph G. If V satisfies the resource constraints described in (1), it serves as the initial partition V^* for the next step. Conversely, we apply a Node Resource Fine-tuning Algorithm to obtain an initial partition V^* that satisfies the constraints.

In the Node Resource Fine-tuning Algorithm, a specially designed min-heap q is used to store all nodes with a specific violated resource k within the subset V_o. The elements in the heap are sorted according to the node movement gain. The algorithm repeatedly takes the top element from min-heap q, adjusts its partition, and updates q until all subsets meet resource constraints. This ensures an effective, nearly equal-sized partition that satisfies multi-dimensional resource constraints and offers superior partition quality.

B. Supernode-Level Partitioning

This section enhances the partition by moving nodes in batches under constraints to achieve higher quality. Moving nodes in batches leverages global information, significantly improving partition quality. This approach also reduces graph scale, decreasing algorithm time overhead.

Specifically, we reapply the balanced partitioning algorithm within the initial partition V, coarsening the nodes in the graph to form supernodes. A supernode is formed by clustering a batch of nodes and treating them as a single node. As a result, the graph is transformed into a supernode graph G_s. The number of supernodes $s \in S$ is determined by fine-grained parameters $P_g \in \mathbb{Z}^+$.

Subsequently, we apply a Supernodes Transferring Algorithm to move the partitions of supernodes. The main process is described as follows:

a) step 1: Initialize information by updating the node information of all boundary supernodes in S. An ordered map named $Candidates$ is maintained to track potential supernodes for improvement after transfer, with values including the supernode's gain and a target subset queue $queue_s$. The elements in $Candidates$ are ordered based on the gain. $queue_s$ is a priority queue that records subsets with stronger connectivity to the supernode.

b) step 2: If $Candidates$ is empty, proceed to step 3. Otherwise, extract the supernode s_{first} with the highest gain from $Candidates$. Try transferring s_{first} to the top subset S_{top} in $queue_s$. If S_{top} lacks resources, keep trying with the next subset in $queue_s$. If $queue_s$ is empty, the information of s_{first} is removed from the $Candidates$, and a secondary candidate supernodes ordered map $Candidates'$, is utilized to record it. Conversely, if S_{top} has enough resources, transfer s_{first} successfully and remove s_{first} from $Candidates$. This may relax constraints, so retry transferring all remaining supernodes in the $Candidates'$. Then, repeat step 2.

c) step 3: Supernodes from $candidates'$ are reattempted to be transferred. Following this, the adjusted supernodes partition S' are then output.

After applying this algorithm, we obtained an unbalanced partition S' that satisfies the constraints, with further reductions in Cut and Cut_{max}. Finally, the adjusted supernode graph S' is mapped back to the original node graph V'.

C. Iterative Partitioning

The aforementioned Supernode-Level Partitioning is iterated multiple times until the quality converges. In a single process, the number of coarsened supernodes directly determines the quantity of batched nodes moved in each operation. This is governed by the fine-grained parameter $P_g \in \mathbb{Z}^+$. P_g is initialized to 1, treating the entire subset as a single supernode. Then, P_g is updated according to the following (5):

$$P_{g_{\text{next}}} = \lceil \alpha \cdot P_g \rceil, \quad \alpha > 1. \tag{5}$$

In (5), the symbols $\lfloor \cdot \rfloor$ represent the ceiling function. The design of (5) is specifically intended to prioritize and focus on moving larger batches.

During the Supernode-Level Partitioning process, the size of supernodes gets smaller over time. If the partition quality doesn't improve, we reset and restart. The algorithm halts and returns the final partition when the partition quality stabilizes.

D. Effectiveness Analysis

Our proposed FPGA-Par method is inspired by the concept of $a-adic\ expansion$ from number theory. Partition's size is denoted by $|A|$, the optimal partition reduces this to $|A| - |A'|$, where $|A'|$ represents the size of nodes moved outside the partition to achieve a smaller Cut. According to the $a-adic\ expansion$, their relationship can be expressed as an infinite series in (6):

$$\frac{|A'|}{|A|} = b_0 \left(\frac{1}{\alpha}\right)^0 + b_1 \left(\frac{1}{\alpha}\right)^1 + b_2 \left(\frac{1}{\alpha}\right)^2 + \ldots \leq 1, \quad b_i \in \{0,1\}, \quad \alpha > 1. \tag{6}$$

Approximately, by taking the first l terms and ensuring each term is an integer, it can be transformed into (7):

$$|A'| = \left(b_0 \lfloor \left(\frac{1}{\alpha}\right)^0 \rfloor + b_1 \lfloor \left(\frac{1}{\alpha}\right)^1 \rfloor + \ldots + b_l \lfloor \left(\frac{1}{\alpha}\right)^l \rfloor \right) \cdot |A|. \tag{7}$$

The base α corresponds to the fine-grained parameter p_g, then (7) indicates that under the ideal balanced partitioning for coarsening, a subset $|A|$ of the original partition can be iteratively divided into smaller subregions of size $|A'|$, achieving a higher quality partition.

Furthermore, based on (6) and (7), the relationship between the given $|A|$ and $|A'|$ is as illustrated in (8):

$$|A'| = \left(\sum_{i=0}^{l} b_{\alpha_i} \left\lfloor \left(\frac{1}{\alpha}\right)^i \right\rfloor + \sum_{i=0}^{m} b_{\beta_i} \left\lfloor \left(\frac{1}{\beta}\right)^i \right\rfloor + \sum_{i=0}^{n} b_{\gamma_i} \left\lfloor \left(\frac{1}{\gamma}\right)^i \right\rfloor + \ldots \right) \cdot |A|, \quad \alpha_i, \beta_i, \gamma_i \in \{0,1\}, \quad \alpha, \beta, \gamma > 1. \tag{8}$$

In (8), the bases α, β and γ correspond to different fine-grained parameters p_g that are updated throughout the Iterative Partitioning process. This implies that when coarsening is suboptimal, leading to Supernode-Level Partitionings under the same p_g that fail to meet the ideal conditions specified in (7), the desired partition sizes $|A'|$ of subset $|A|$ can still be achieved by Iterative Partitioning using multiple different p_g, as presented in the form of (8).

Based on the formulaic analysis above, consider also that a subset $|A|$ of the original partition needs to be divided into a subinterval $|A'|$. The complexity of node movements required by our algorithm is $O(\log |A'|)$, in contrast to $O(|A'|)$ for traditional methods that rely on single-node movements.

TABLE I: Statistics of S2C Benchmarks and Performance of Different Imbalanced Partitioning Algorithms.

Benchmark	*Nodes*	*Parts*	Kahypar (Exhaustive)			Kahypar (Ternary Search)			FPGA-Par		
			Cut	Cut_{max}	Time(s)	*Cut*	Cut_{max}	Time(s)	*Cut*	Cut_{max}	Time(s)
case1	71	2	4	4	0.44	4	4	0.24	6	6	0.59
		8	52	22	5.44	8	3	0.78	4	4	0.61
		20	241	8	7.02	16	3	1.52	5	5	0.62
case2	450	2	180	180	0.31	178	178	0.78	178	178	3.03
		8	212	64	18.3	210	71	3.93	208	50	3.11
		20	885	66	34.95	245	70	7.7	231	35	3.03
case3	5084	2	2256	2256	6.07	2303	2303	17.93	2665	2665	19.31
		8	2717	899	88.94	3165	1132	23.87	3601	920	21.9
		20	9924	457	172.65	2639	1092	41.01	3305	428	23.91
case4	54961	5	24886	5759	264.35	23001	6598	613.92	24886	5759	377.06
		8	26125	4965	910.47	25612	5736	423.58	22310	5452	365.59
		20	49013	4045	2177.98	32132	4853	699.27	23567	3019	415.4
case5	62753	5	21385	8363	265	19459	8980	687.59	19068	7363	263
		8	27712	5166	957.35	27838	5456	1209.42	30251	6288	275.26
		20	42866	2387	3696.71	31567	5506	897.68	18942	2339	359.04
case6	127086	5	47005	11856	10634.66	43361	14381	9092.48	6330	6330	3621.01
		8	57068	8901	23415.09	52112	9677	12405.89	6867	6867	4420.05
		20	68035	3863	55588.5	5133	2294	4320.89	5863	3500	3765.95
Average Ratio			1	1	1	0.71	1.15	0.31	0.44	0.86	0.14

IV. Experiment

We implemented our proposed FPGA-Par in Python an tested it on a 64-bit Linux workstation with an Intel Xeon Gold 6326 2.9GHz CPU with 503GiB memory, all within the constraints of a single thread. The benchmark are provided by S2C Ltd. [17], encompassing 9 types of resource constraints.

Table I shows how our FPGA-Par algorithm compares to the state-of-the-art KaHyPar [7] algorithm for unbalanced partitioning. For exploring the imbalance factors in KaHyPar, we tried two methods: one that exhaustively checks every possible imbalance factor starting from 0.03 and going up, and another that uses ternary search to find the best partition. Our FPGA-Par uses KaHyPar's balanced partitioning for part of its process. The metrics used to evaluate the quality of graph partitioning in Table I are Cut and Cut_{max}, as described by equations (3) and (4), respectively.

Table I demonstrates that the exhaustive exploration of the imbalance factors in KaHyPar is time-consuming, whereas using ternary search reduces the time to 31%. Interestingly, while the Cut_{max} is slightly higher with the ternary search than with the exhaustive method, a slightly relaxed Cut_{max} results in a significant reduction in Cut. In our proposed method, FPGA-Par, the time consumed is further reduced to 14% compared to the exhaustive KaHyPar, while maintaining superior graph partitioning quality: the average reduction in Cut is 66%, and in Cut_{max}, 14%. Since FPGA-Par does not require exploration of the imbalance factors, its time overhead is significantly reduced. In the implementation of the FPGA-Par algorithm, iterative movements of supernodes involve moving batches of nodes at a time, leading to significant improvements in the overall quality of the graph.

For each case in Table I, we selected the best partition obtained by each method, and then routed and mapped these partitions onto a 5x5 multi-FPGA system. The system frequency was calculated according to (1), with the results shown in Table II.

TABLE II: System Frequency After Mapping and Routing to a 5x5 Mesh Multi-FPGA System with Different Methods.

Benchmark	Kahypar (Exhaustive)		Kahypar (Ternary Search)		FPGA-Par	
	freq	Ratio	*freq*	Ratio	*freq*	Ratio
case1	2500	1	2500	1	2500	1
case2	26.6	1.41	18.84	1	25.23	1.34
case3	4.05	0.53	7.7	1	9.45	1.23
case4	1.05	0.59	1.78	1	1.85	1.04
case5	1.66	0.99	1.68	1	2.64	1.57
case6	0.497	0.83	5.97	1	7.89	1.32
Average Ratio		0.77		1		1.25

Table II indicate that the system frequency of partitions obtained using ternary search-based KaHyPar is higher when deployed on a multi-FPGA system compared to those obtained using exhaustive search-based KaHyPar. Compared to the best baseline method, FPGA-Par achieved an average 1.25× improvement in system frequency.

V. Conclusion

In this work, we proposed FPGA-Par, an adaptive graph partitioning algorithm for multi-FPGA systems. Our proposed method transforms the problem of imbalanced partitioning into an iterative balanced partitioning problem. By iteratively coarsening supernodes across multiple granularities and adjusting their placement, it enables the derivation of partitioning solutions with flexible size ratios. The effectiveness of our method has been validated both theoretically and experimentally, demonstrating that our FPGA-Par surpasses the existing state-of-the-art methods.

REFERENCES

[1] Q. Tang, H. Mehrez and M. Tuna, "Routing Algorithm for Multi-FPGA Based Systems Using Multi-Point Physical Tracks," in *Proc. RSP*, 2013, pp. 2-8.

[2] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *The Bell System Technical Journal*, vol. 49, no. 2, pp. 291-307, 1970.

[3] C. M. Fiduccia and R. M. Mattheyses, "A Linear-Time Heuristic for Improving Network Partitions," in *Proc. DAC*, 1982, pp. 175-181.

[4] J. Kim, I. Hwang, Y.-H. Kim, and B.-R. Moon, "Genetic Approaches for Graph Partitioning: A Survey", in *Proc. GECCO*, 2011, pp.473–480.

[5] A. Nazi, W. Hang, A. Goldie, S. Ravi, and A. Mirhoseini, "A Deep Learning Framework for Graph Partitioning," in *Proc. ICLR*, 2019.

[6] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel Hypergraph Partitioning: Applications in VLSI Domain," in *Proc. DAC*, 1997, pp. 526–529.

[7] S. Schlag, V. Henne, T. Heuer, H. Meyerhenke, P. Sanders, and C. Schulz, "k-way Hypergraph Partitioning via n-Level Recursive Bisection," in *Proc. ALENEX*, 2016, pp.53–67.

[8] Y. Shen and G. Zeng, "An Unbalanced Partitioning Scheme for Graph in Heterogeneous Computing," in *Proc. GCC*, 2005, pp. 1167–1172.

[9] Dutt S and Theny H, "Partitioning around roadblocks: tackling constraints with intermediate relaxations," in *Proc. IEEE/ICCAD*, 1997, pp. 350-355.

[10] D. Tummalapalli, C. Kunapareddy, V. Akalwadi, R. Govindan, and B. G., "Novel Design Partitioning Technique for ASIC Prototyping on Multi-FPGA Platforms Using Graph Deep Learning," in *Proc. ICECS*, 2022, pp. 1–4.

[11] Z. Lei, Z. Xiao, Y. Qin, and K. Li, "A Multi-Dimensional Weight Partition Algorithm for FPGA Prototype Simulation," in *Proc. ISEDA*, 2023, pp. 106–109.

[12] D. Zheng, X. Zang, and M. D. F. Wong, "TopoPart: A Multi-level Topology-Driven Partitioning Framework for Multi-FPGA Systems," in *Proc. ICCAD*, 2021, pp. 1–8.

[13] H. Gao, "FPGA-Par: an Adaptive Graph Partitioning Algorithm for Multi-FPGA," https://github.com/HengyueGao/FPGA-Par, 2025.

[14] K. Bouaziz, S. Chtourou, Z. Marrakchi, A. M. Obeid, and M. Abid, "Mesh of Trees FPGA Architecture: Exploration and Optimization," *IEEE TCAD*, vol. 41, no. 9, pp. 2943-2956, 2022.

[15] U. Farooq, R. Chotin-Avot, M. Azeem, Z. Cherif, M. Ravoson, S. Khan, and H. Mehrez, "Using Timing-Driven Inter-FPGA Routing for Multi-FPGA Prototyping Exploration," in *Proc. DSD*, 2016, pp. 641–645.

[16] M. Turki, H. Mehrez, Z. Marrakchi, and M. Abid, "Partitioning Constraints and Signal Routing Approach for Multi-FPGA Prototyping Platform," in *Proc. SoC*, 2013, pp. 1–4.

[17] H. Gao, "S2C Benchmark: Circuit Topologies with Multi-dimensional Resources and FPGA Constraints," https://github.com/HengyueGao/S2C_Benchmark, 2025.