Late Breaking Results: Hybrid Logic Optimization with Predictive Self-Supervision

Rongliang Fu[1], Ran Zhang[2], Ziyang Zheng[1], Zhengyuan Shi[1], Yuan Pu[1], Junying Huang[2*], Qiang Xu[1] and Tsung-Yi Ho[1]
[1]*Department of Computer Science and Engineering, The Chinese University of Hong Kong*, Hong Kong, China
[2]*State Key Lab of Processors, Institute of Computing Technology, CAS*, Beijing, China

***Abstract*—Hybrid optimization is an emerging approach in logic synthesis, focusing on applying diverse optimization methods to different parts of a logic circuit. This paper analyzes the relationship between each vertex and its corresponding optimization method. We extract a subgraph centered on each vertex and quantify the logic optimization results of these subgraphs as vertex features. Based on these features, we propose a circuit partitioning method to cluster the logic circuit, enabling the final optimized circuit to be constructed by merging clusters optimized with their respective methods. Additionally, we introduce a self-supervised prediction model to efficiently obtain vertex features. The experimental results targeting LUT mapping demonstrate that our method achieves improvements of 8.48% in area and 9.81% in delay compared to the state-of-the-art.**

I. Introduction

Logic synthesis plays a crucial role in contemporary electronic design automation (EDA) [1], involving logic optimization and technology mapping. Early efforts focused on simplifying Boolean expressions using algebraic techniques. As the complexity of digital circuits increased, the need for more advanced optimization techniques led to the development of multi-level logic optimization [2], which considers entire networks of logic gates rather than isolated expressions. The introduction of directed acyclic graphs (DAGs) for representing Boolean networks enabled more effective logic optimization by facilitating the exploration of different structural configurations. Various types of graphs, such as And-Inverter graphs (AIGs) [3], Majority-Inverter graphs (MIGs) [4], Xor-And-Inverter graphs (XAGs) [5], and Xor-Majority-Inverter graphs (XMGs) [6], were developed to capture different logic functions and enable algebraic manipulation.

The introduction of these graphs has facilitated the development of numerous logic optimization techniques, such as rewriting, resynthesis, and resubstitution. However, these methods mainly focus on a single type of graph representation. Due to the complexity of graph structures, hybrid optimization, a combination of these optimization techniques, has the potential to significantly enhance circuit performance. Neto et al. proposed LSOracle [7], which seeks to identify different clusters within a given circuit and assigns a circuit graph optimization technique to each cluster to explore a better circuit design. Although LSOracle has demonstrated promising results in reducing the area-delay product (ADP), it has notable limitations: (i) it directly uses a general graph partition tool, KaHyPar [8], to partition the circuit, without accounting for unique characteristics of circuit design, and (ii) it employs a neural network model to predict the optimization method of each cluster, which may underperform on complex circuits.

* Corresponding author: huangjunying@ict.ac.cn.

Algorithm 1: Circuit partition

Input: Boolean network $G(V, E)$ and param.: α, β, λ.
Output: Partition results cs.

```
depth, size ← Get the depth and size of G
φ = 10^(10×depth/size)
while size > λ do
    ζ ← Determine the block depth
    Γ ← Partition graph G into blocks of depth ζ
    cs ← {}
    #block − based parallel running
    for γ ∈ Γ do
        g ← Leiden{γ}, and then remove cycles in g
        cs ← cs ∪ {g}
    #end parallel running
    G ← Re-construct a graph based on clusters cs
    depth, size ← Get the depth and size of G
if φ ∈ (α, β) then
    Merge vertices connected with large closeness in G
    cs ← Obtain the clusters from vertices within G
else
    cs ← Girvan-Newman{G}
return cs
```

To address these limitations and incorporate circuit-specific characteristics in hybrid optimization, this paper proposes a novel hybrid optimization framework using a self-supervised prediction model. The framework extracts physical-aware features of each vertex in a given Boolean network and then uses these features to partition the network and identify the most suitable optimization strategy for each cluster. The experimental results on the EPFL benchmark [9] show the effectiveness of our framework.

II. Problem Formulation

Hybrid logic optimization involves the application of various circuit graphs and corresponding logic optimization methods to a Boolean network. This approach aims to minimize both delay and area. The problem can be formally defined as:

- Input: A Boolean network $G(V, E)$.
- Output: An optimized logic circuit $G'(V', E')$.
- Goal: $\min\{\text{ADP} = \text{delay} \times \text{area}\}$, which can facilitate a more effective balance between delay and area.

III. Hybrid Optimization Framework

A. Labeling Vertices

To label vertices in a Boolean network, we extract features from an expanded subgraph $g_k(v)$ around each vertex v, defined by a partial order $\preccurlyeq_k$. The costs of n optimization methods applied to $g_k(v)$ are represented as a vector $\text{Cost}(v) = [\text{ADP}_1, \text{ADP}_2, \cdots, \text{ADP}_n]$, which is softmax-normalized to produce the label vector $L(v)$, guiding partition and optimization tasks. For efficient prediction of $L(v)$, we use DeepGate3 [11] to generate functional (HF) and

TABLE I: Experimental results on the EPFL benchmark [9] after 6-LUT mapping, where $\{\alpha, \beta, \lambda\} = \{7, 9, 15\}$.

Circuit	AIG [10]			MIG [10]			XMG [10]			XAG [10]			LSOracle [7]			Ours		
	delay	area	ADP	delay	area	ADP	delay	area	ADP	delay	area	ADP	delay	area	ADP	delay	area	ADP
Voter	29	2013	1.27	27	2372	1.39	32	2213	1.54	34	3328	2.46	28	2001	1.22	25	1841	1
Adder	64	192	2.81	11	397	1.00	14	409	1.31	64	192	2.81	11	397	1.00	11	397	1
Bar	4	512	1.00	4	512	1.00	4	512	1.00	4	512	1.00	4	512	1.00	4	512	1
Log2	182	7889	1.21	135	9002	1.02	192	7796	1.26	185	8125	1.26	152	8366	1.07	131	9093	1
Arbiter	9	1876	1.92	9	2458	2.51	9	2726	2.78	11	1921	2.40	11	1912	2.39	5	1763	1
Priority	35	225	1.00	33	264	1.11	33	265	1.11	33	265	1.11	35	225	1.00	35	225	1
Multiplier	127	6356	1.78	68	7453	1.12	75	7504	1.24	119	6907	1.82	92	6800	1.38	75	6034	1
Cavlc	8	114	1.28	6	119	1.00	6	119	1.00	6	119	1.00	8	114	1.28	6	119	1
I2c	5	342	1.15	5	342	1.15	5	342	1.15	5	342	1.15	5	362	1.22	5	297	1
Sin	91	1587	1.16	64	1929	0.99	83	1583	1.05	83	1583	1.05	77	1741	1.07	71	1759	1
Div	1996	4231	1.00	732	28889	2.50	762	26647	2.40	1606	25529	4.85	2006	9491	2.25	1996	4231	1
Ctrl	2	28	1.00	2	28	1.00	2	28	1.00	2	28	1.00	2	28	1.00	2	28	1
Square	48	4992	1.64	30	4822	0.99	122	4043	3.38	122	4130	3.45	55	4454	1.68	32	4559	1
Dec	2	273	1.00	2	273	1.00	2	273	1.00	2	273	1.00	2	273	1.00	2	273	1
Router	13	53	1.00	13	53	1.00	13	53	1.00	13	53	1.00	10	68	0.99	13	53	1
Max	45	1689	3.15	22	1529	1.39	22	2288	2.09	89	787	2.90	45	1689	3.15	24	1005	1
Memctrl	56	12326	1.35	41	13309	1.07	45	11969	1.05	49	11709	1.12	46	12164	1.09	43	11913	1
Int2float	4	47	1.00	4	47	1.00	4	47	1.00	4	47	1.00	4	56	1.19	4	47	1
Sqrt	2004	4045	1.06	3386	8044	3.55	3386	8044	3.55	3689	7839	3.77	1965	3924	1.01	1960	3912	1
Hyp	8520	49736	1.00	8520	49736	1.00	8520	49736	1.00	8520	49736	1.00	8525	51943	1.05	8521	49530	1
Ave. ratio	1.46	1.02	1.39	1.03	1.45	1.34	1.24	1.43	1.55	1.69	1.32	1.86	1.18	1.14	1.35	1	1	1

structural (HS) embeddings for $g_k(v)$. These embeddings are concatenated and processed by a Multi-Layer Perceptron to predict $\hat{L}(v)$. The loss is computed by comparing the softmax-normalized ground truth $L(v)$ and predicted $\hat{L}(v)$.

B. Circuit Partition

We propose a Boolean network partition method combining hierarchical partitioning and re-clustering to address the challenges in Section II, as detailed in Algorithm 1. The hierarchical partition iteratively divides the network into blocks based on logic depth ζ (lines 4-5), guided by a measure φ that balances depth and size (line 2). Blocks are further partitioned into clusters using the Leiden method [12], with edge weights redefined to emphasize vertex optimization similarity (lines 7-11). Cycles between clusters are resolved through merging (line 9), and a new graph is constructed with clusters as vertices (line 12), repeating the process until the graph size meets a threshold λ (lines 3-13). To enhance global partition quality, re-clustering is applied based on φ: for balanced networks ($\varphi \in (\alpha, \beta)$), vertices with high closeness are merged (lines 14-16), while for imbalanced networks, the Girvan-Newman algorithm [13] is used to refine clusters (line 18).

C. Sub-circuit Optimization

After partitioning the Boolean network into clusters, the optimization method for each cluster $c \in cs$ is determined based on the tendency $T(c)$, calculated by summing the one-hot encoded labels $\Phi(L(v))$ of its vertices, where $\Phi(L(v))$ highlights the preferred optimization method for vertex v. The final optimization method for each cluster is selected via $\mathrm{argmax}(T(c))$, favoring the method preferred by the majority of vertices. After optimizing all clusters, the optimized clusters are merged to form the final circuit G'.

IV. Experimental Results

All experiments were conducted on a machine with Intel(R) Xeon(R) Platinum 8350C processor, NVIDIA A100 GPU, and 1.5 TB memory. To assess the effectiveness of our framework, we targeted 6-LUT mapping and used the optimization methods corresponding to the four circuit graphs provided by [10], as well as LSOracle [7] as baselines. Training datasets included all aiger files from ISCAS'89 [14] and 5,000 subgraphs from Square [9]. After 400 epochs, the loss converged to 0.0192, with a 96.83% success rate in predicting the minimum value of L. Table I shows that our method outperforms the baselines in terms of area, delay, and ADP, and reduces them by 8.48%, 9.81%, and 17.46% respectively compared to LSOracle.

Acknowledgment

The work was conducted in the JC STEM Lab of Intelligent Design Automation funded by The Hong Kong Jockey Club Charities Trust. It was jointly funded by the Research Grants Council of Hong Kong SAR (No. CUHK14207523) and the National Natural Science Foundation of China (No. 62302477).

References

[1] Testa *et al.*, "Logic synthesis for established and emerging computing," *Proceedings of the IEEE*, vol. 107, no. 1, pp. 165–184, 2019.

[2] Brayton *et al.*, "MIS: A multiple-level logic optimization system," *IEEE TCAD*, vol. 6, no. 6, pp. 1062–1081, 1987.

[3] L. Hellerman, "A catalog of three-variable or-invert and and-invert logical circuits," *IEEE Transactions on Electronic Computers*, vol. EC-12, no. 3, pp. 198–223, 1963.

[4] Amarú *et al.*, "Majority-inverter graph: A novel data-structure and algorithms for efficient logic optimization," in *Proc. DAC*, 2014.

[5] Meuli *et al.*, "Xor-And-Inverter graphs for quantum compilation," *npj Quantum Information*, vol. 8, 12 2022.

[6] Haaswijk *et al.*, "A novel basis for logic rewriting," in *Proc. ASPDAC*, 2017, pp. 151–156.

[7] Neto *et al.*, "LSOracle: a logic synthesis framework driven by artificial intelligence: Invited paper," in *Proc. ICCAD*, 2019.

[8] Schlag *et al.*, "High-quality hypergraph partitioning," *ACM Journal of Experimental Algorithmics*, vol. 27, 2023.

[9] L. Amarù, P.-E. Gaillardon, and G. De Micheli, "The EPFL combinational benchmark suite," in *Proc. IWLS*, 2015.

[10] Soeken *et al.*, "The EPFL logic synthesis libraries," 2022.

[11] Shi *et al.*, "DeepGate3: Towards scalable circuit representation learning," *arXiv preprint arXiv:2407.11095*, 2024.

[12] Traag *et al.*, "From Louvain to Leiden: guaranteeing well-connected communities," *Scientific Reports*, vol. 9, no. 5233, pp. 2045–2322, 2019.

[13] Despalatović *et al.*, "Community structure in networks: Girvan-Newman algorithm improvement," in *Proc. MIPRO*, 2014, pp. 997–1002.

[14] Hansen *et al.*, "Unveiling the ISCAS-85 benchmarks: A case study in reverse engineering," *IEEE Des. Test*, vol. 16, no. 3, pp. 72–80, 1999.