Efficient Approximate Logic Synthesis with Dual-Phase Iterative Framework

Ruicheng Dai[1], Xuan Wang[1], Wenhui Liang[1], Xiaolong Shen[2], Menghui Xu[2],
Leibin Ni[2], Gezi Li[2], and Weikang Qian[1,3]
[1]UM-SJTU Joint Inst. and [3]MoE Key Lab of AI, Shanghai Jiao Tong University, China; [2]Huawei Tech. Co., Ltd., China

***Abstract*—Approximate computing is an emerging paradigm to improve the energy efficiency for error-tolerant applications. Many iterative approximate logic synthesis (ALS) methods were proposed to automatically design approximate circuits. However, as the sizes of circuits grow, the runtime of ALS grows rapidly. Thus, a crucial challenge is to ensure circuit quality while improving the efficiency of ALS. This work proposes a dual-phase iterative framework to accelerate the iterative ALS flows. In the first phase, a comprehensive circuit analysis is performed to gather the necessary information, including the error information. In the second phase, minimal incremental computation is employed based on the information from the first phase. The experimental results show that the proposed method achieves an acceleration by up to $21.8\times$ without loss of circuit quality compared to the state-of-the-art methods.**

***Index Terms*—approximate logic synthesis, error estimation, large-scale circuit synthesis**

I. Introduction

With the advent of the post-Moore era, reducing circuit costs becomes more challenging [1]. Approximate computing is a novel energy-efficient design paradigm for error-tolerant applications such as machine learning, data mining, and image processing [2]–[6]. It intentionally incorporates a slight inaccuracy in computation, which can significantly decrease the circuit area, delay, and power consumption.

At the circuit level, approximate computing is realized through designing approximate circuits [7]. One approach to designing such circuits is through *approximate logic synthesis (ALS)* [8], which automatically synthesizes an optimized circuit while satisfying a given error constraint. Recently, iterative ALS methods [9]–[19] have gained popularity due to their superior synthesis quality. These methods often apply mutiple rounds of *local approximate changes (LACs)* to obtain the final approximate circuit, where LACs involve replacing a node in a certain way to simplify a sub-circuit. For example, in Fig. 1, a SASIMI [13] LAC replaces node b with another existing node a to simplify the circuit. For these iterative ALS methods, each iteration involves identifying and evaluating all candidate LACs of the current approximate circuit, and the evaluation typically involves evaluating the errors of all candidate LACs. After the evaluation, the best LACs are selected and applied to the current circuit to generate a new approximate circuit.

However, as the sizes of the circuits increase, the runtime of the iterative ALS methods grows rapidly, since both the runtime to accurately estimate the errors of all candidate LACs [19] and the number of iterations increase. For example, more than 22 hours are needed to synthesize a large circuit in the work [12]. To address the above issue, some works reduce the number of iterations by selecting multiple LACs in each iteration [14]–[16]. However, the works [14] and [15] sacrifice the accuracy of error estimation since it is hard to predict the error caused by multiple LACs without simulating the whole circuit, which may eventually degrade the synthesis quality, and the work [16] employs exact synthesis method to generate candidate LACs, which is time-consuming. Other works accelerate the error estimation by avoiding the repeated computation in one iteration [19], [20], which still iteratively select a single LAC in each iteration to guarantee accurate error estimation. However, these works ignore the fact that there exists much repeated computation for circuit analysis in a few consecutive iterations. For example, in Fig. 1, after applying the SASIMI LAC, the works [19], [20] will evaluate all nodes of the new approximate circuit to update some structure information needed to perform accurate error estimation. In fact, for the two circuits before and after applying the LAC, most of the nodes remain unchanged, and there exist identical sub-circuits. Indeed, we only need to update the structure information of two nodes, a and c.

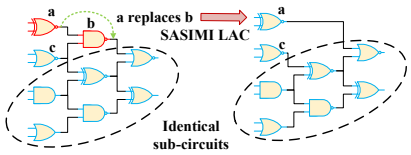


Fig. 1. Example of SASIMI LAC.

Inspired by the above idea, this work proposes a dual-phase iterative framework to further accelerate the iterative ALS flows while preserving the accuracy of the error estimation. In the first phase of the framework, a comprehensive circuit analysis is performed to gather the necessary information, including the error information. In the second phase, minimal incremental computation is employed based on the information from the first phase. To guarantee the accuracy of error estimation, this work still selects a single LAC in each iteration. The main contributions are listed as follows.

1) This work proposes a novel dual-phase iterative ALS framework. With comprehensive analysis in the first phase, it achieves significant acceleration in the second phase by selecting the most promising LACs through incremental computing.

This work is supported by the National Natural Science Foundation of China under Grants T2293700, T2293701, and T2293704. First author: Ruicheng Dai (email: ruichengdai@sjtu.edu.cn). Corresponding author: Weikang Qian (email: qianwk@sjtu.edu.cn).

2) This work proposes an incremental method to efficiently obtain the cuts that are necessary to perform accurate error estimation.
3) This work proposes a method to only perform essential computation for the most promising LACs to accelerate the error estimation.
4) To ensure good performance for different circuits, we propose an adaptive method to tune the parameters to accelerate the most time-consuming step and conditions to start a new dual phase for higher circuit quality.

Our method can be applied to any input distribution and any statistical error metric. In this work, we consider three statistical error metrics, *error rate (ER)*, *mean square error (MSE)*, and *mean error distance (MED)*. ER measures the proportion of incorrect outputs. MSE measures the average of the squares of the errors. MED measures the average error distance. The experimental results show that our method outperforms the state-of-the-art methods in runtime. For example, our method is $9.0\times$ faster for small circuits and $21.8\times$ faster for large circuits than an enhanced version of the method in [19].

II. Preliminaries

A. Circuit Terminologies

A multi-level combinational circuit can be modeled as a directed acyclic graph (DAG). The primary inputs (PIs) are a subset of the nodes in the graph without any fanins, and the primary outputs (POs) are another subset of the nodes without any fanouts. In a circuit, if there is a path from node a to node b, then b is a transitive fanout (TFO) of a, and a is a transitive fanin (TFI) of b. The TFO (*resp.*, TFI) cone of a is a set of nodes including a and all TFOs (*resp.*, TFIs) of a.

B. Error Estimation for ALS

As mentioned in Section I, an iterative ALS method typically needs to evaluate the errors of all candidate LACs. This section introduces the state-of-the-art methods for accurate and efficient error estimation used in ALS.

A representative method is a batch error estimation method called VECBEE based on Monte Carlo simulation [19]. The main idea is to calculate the error increases of all LACs by a change propagation matrix (CPM). A CPM P for a circuit is a 3-dimensional 0-1 matrix of size $X \times Y \times O$, where X is the number of input patterns for simulation, Y is the node number of this circuit, and O is the number of POs. Each entry $P[i, n, o]$ is a Boolean difference of a PO o with respect to a node n, representing whether a value change on node n can be propagated to PO o under the i-th input pattern. If so, $P[i, n, o] = 1$. Otherwise, $P[i, n, o] = 0$. With CPM, we can efficiently calculate the error increases of all candidate LACs, since we only need to obtain under which input patterns the value of a node changes after applying a LAC, and then the changes on POs are obtained immediately using the CPM.

To efficiently and accurately obtain CPM, the work [19] introduces some auxiliary Boolean difference entries $P[i, n, s]$ where s can be any node in the circuit. Further, it defines a *one-cut* for a node n and a PO o as a node t satisfying that all paths from n to o pass through t. For example, in Fig. 2, for node a and PO O_2, a one-cut can be e since there are two paths $a \to b \to e \to O_2$ and $a \to c \to e \to O_2$ from a to O_2, and they both pass e.

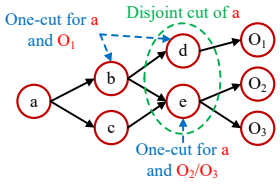


Fig. 2. Example of one-cut and disjoint cut.

A one-cut node t divides the sub-circuit between n and o into two sub-circuits. The first is the sub-circuit between n and t, and the second is the one between t and o. If a change on n can propagate to o under the i-th input pattern (*i.e.*, $P[i, n, o] = 1$), then it should first propagate to t in the first sub-circuit (*i.e.*, $P[i, n, t] = 1$), and then the value change on t should further propagate to o in the second sub-circuit (*i.e.*, $P[i, t, o] = 1$). Thus, we have the following equation:

$$P[i, n, o] = P[i, t, o] \wedge P[i, n, t]. \tag{1}$$

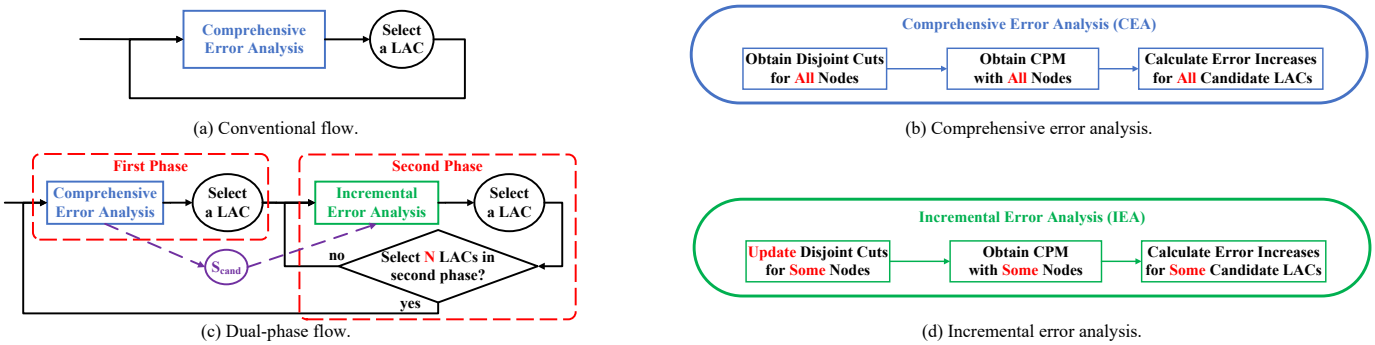
Eq. (1) shows that once we know $P[i, t, o]$, we can get $P[i, n, o]$, where node t, as a one-cut of node n and PO o, is a TFO of node n. Thus, we can obtain all entries in the CPM by applying Eq. (1) in a reverse topological order over all nodes in the circuit.

Another similar work for error estimation is proposed in [20], which introduces the *closest disjoint cut*. A *disjoint cut* of node n is a set of one-cuts for n and all POs reachable from n such that 1) exactly one one-cut is selected for each reachable PO, and 2) TFOs of all nodes in this set do not intersect with each other. For example, in Fig. 2, one disjoint cut of a is $\{d, e\}$, since the reachable POs from a are O_1, O_2, and O_3. For a and O_1 (*resp.*, O_2 and O_3), there exists one one-cut, *i.e.*, node d (*resp.*, node e), in this set, and the TFOs of d and e have no intersection. The work [20] proposes an algorithm to find the disjoint cut of node n that is closest to n, which is called *the closest disjoint cut*, denoted as $C_{cd}(n)$. For convenience, the term disjoint cut in the following discussion generally refers to the closest disjoint cut. Compared to one-cut, an advantage of disjoint cut is that only one simulation is needed for node n to obtain the Boolean differences $P[i, n, f]$ for all nodes f in the cut, while in VECBEE, for each node n, each PO requires a separate simulation based on its algorithm. Thus, calculating CPM based on disjoint cut is usually more efficient than one-cut.

III. Methodology

A. Overview of Proposed Method

This section overviews the proposed dual-phase ALS framework. First, we present a conventional ALS flow, which is shown in Fig. 3(a). To ensure the quality of the output approximate circuit, the conventional flow selects a single LAC in each iteration. To decide which LAC to select, in each iteration, the conventional flow performs a comprehensive error



(a) Conventional flow.

(b) Comprehensive error analysis.

(c) Dual-phase flow.

(d) Incremental error analysis.

Fig. 3. A conventional ALS flow and our proposed ALS flow.

analysis, which applies the efficient accurate error estimation method proposed in [19], but it replaces the one-cut in [19] by the disjoint cut proposed in [20], since Section II-B shows that the latter is more efficient for obtaining the CPM. Fig. 3(b) shows the details of the comprehensive error analysis, which has three steps: 1) obtaining the disjoint cuts, 2) calculating the CPM, and 3) calculating the error increases for all LACs.

In this work, we only consider LACs whose affected local circuits have a single output. We call the single output node the *target node* of a LAC (*e.g.*, node b in Fig. 1). In what follows, when a LAC is applied, we also say that the target node of the LAC is replaced. For node a, let $E(a)$ be the smallest error increase among the error increases of all LACs whose target node is a. Our basic motivation is inspired by the experimental result shown in Fig. 4. In this experiment, for each circuit, we perform the conventional flow to analyze all possible LACs, and each iteration selects one target node with the smallest error increase to be replaced. We call the target node the *optimal choice*. At the end of the first iteration, we sort all remaining nodes n by $E(n)$ in ascending order and select the top 60 nodes to form a candidate node set S. Then, we obtain the optimal choices selected in the next k rounds and count how many of them are also in S. We denote the number by T_k and calculate the percentage T_k/k.

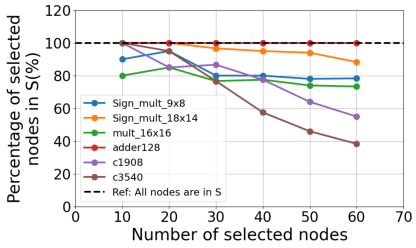


Fig. 4. Experiment on candidate node set selection.

We plot T_k/k in Fig. 4 for some benchmarks for $k = 10, 20, \ldots, 60$. The result shows that most optimal choices in subsequent iterations are still in S. Specifically, in the next 10, 20, and 30 iterations, more than 80% of optimal choices are in S for almost all tested circuits. The result is reasonable since $E(a)$ reflects a's contribution to the output to some extent. For example, if node a only affects some POs of lower weights, and b affects some POs of higher weights, $E(a)$ is usually smaller than $E(b)$ no matter how other nodes are replaced. Thus, to improve the efficiency, a proper attempt is to only consider replacing the nodes in the candidate node set S instead of all nodes, since the optimal choice of an iteration is very likely in the candidate node set. However, some optimal choices in some subsequent iterations may not be in the set. To guarantee that the candidate node set contains the optimal choices, we need to update the set after some iterations.

Motivated by the above observations, we propose a dual-phase framework as shown in Fig. 3(c). In each iteration, it has two phases. The first phase performs a comprehensive error analysis, which is the same as the conventional flow. It also selects a candidate node set of size M with the smallest error increases, denoted as S_{cand}, for the second phase. The second phase performs incremental error analysis with an iteration limit N. In each round of the second phase, we only consider replacing a node in the set S_{cand} instead of in the whole set of nodes in the circuit, thus accelerating the ALS flow. Note that we set $N < M$, since it is unlikely that all nodes in S_{cand} are optimal choices. Fig. 3(d) shows the details of the incremental error analysis. It has three steps corresponding to the three steps in the comprehensive error analysis. The first step is an accelerated version of the one in the comprehensive error analysis. After applying a LAC, only a few nodes need to update their disjoint cuts, such as nodes a and c in the new approximate circuit in Fig. 1. Thus, we update the disjoint cuts only for these nodes, and for the others, we reuse their cut information obtained from the first phase (see Section III-B for details). The second step is accelerated by only calculating the CPM entries required to derive those for the nodes in S_{cand}, while the third step is accelerated by performing error estimation only for the LACs that are applied on nodes in S_{cand}. The details of the second and the third steps will be described in Section III-C. To enhance our flow, we also propose self-adaption techniques for the second phase, which will be described in Section III-D.

B. Incremental Update of Disjoint Cut

This section explains the method for the incremental update of disjoint cut. First, we describe how to identify a node n whose previous disjoint cut is no longer a disjoint cut after a LAC is applied. According to the definition of disjoint cut in Section II-B, the disjoint cut of n depends on all paths from node n to all POs reachable from n, which is determined by the structure of n's TFO cone. Therefore, if the structure remains unchanged, the disjoint cut remains the

same. Specifically, if the applied LAC does not 1) add or remove any node in n's TFO cone, or 2) modify any edge between nodes in n's TFO cone, the previous disjoint cut of n is still a disjoint cut. We can directly reuse the cut information of the previous approximate circuit before the LAC is applied. We denote the above two conditions as *cut preservation condition (CPC)* of a node. If the CPC is violated, the disjoint cut may need to be updated, as shown in the following example.

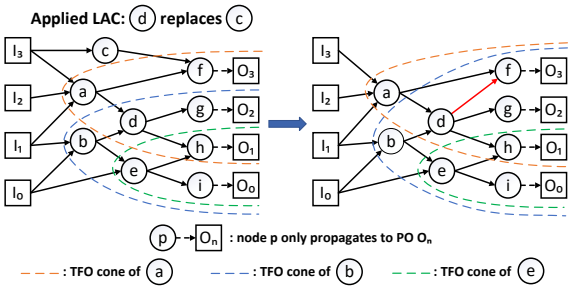


Fig. 5. Example of disjoint cut update after node d replaces node c.

Example 1. *Fig. 5 illustrates the DAGs of an example circuit before and after applying a LAC that replaces node c with node d. After the replacement, node c is removed, and its direct fanout, node f, is transferred to be a fanout of node d. For nodes b and d, their TFO cones now include node f, which violates the first condition. The previous disjoint cut of d (resp., b) is $\{g, h\}$ (resp., $\{g, h, i\}$), and it should be updated to $\{f, g, h\}$ (resp., $\{f, g, h, i\}$). For node a, both d and f are in a's TFO cone, and the additional edge from d to f violates the second condition. As a result, its previous disjoint cut $\{d, f\}$ is no longer a disjoint cut since the TFO cones of d and f intersect. For the remaining nodes (e.g., node e), CPC is satisfied after the replacement, so their disjoint cuts do not need to be updated. Thus, we only need to update the disjoint cuts of nodes a, b, and d.*

It is possible that the CPC is violated but the cut remains a disjoint cut. However, to be conservative, we update all nodes whose CPCs are violated. A property is that if CPC of node n is violated, then it is also violated for any TFI of n, since the TFO cone of a TFI of n includes n's TFO cone. Thus, we first identify the set of all nodes that either change themselves (*i.e.*, are removed or newly created) or change their fanouts (*i.e.*, add or delete any fanout), denoted as S_c. Then, we obtain the set S_v of nodes that violate the CPC by taking the union of all TFI cones of the nodes in S_c and excluding the removed nodes. The cut information of the circuit can be correctly updated if we only update the cuts of nodes in S_v. For example, in Fig. 5, c is removed and d has an additional fanout, so $S_c = \{c, d\}$. Taking the union of c and d's TFI cones and excluding c, we obtain $S_v = \{a, b, d\}$, which is the same as discussed in Example 1.

C. Accelerating Steps Two and Three

This section shows how we accelerate steps two and three in phase two. In the procedure of obtaining the CPM, *i.e.*, step two, the conventional method needs to obtain the Boolean differences $P[i, p, q]$ for all nodes p in the circuit and $q \in C_{cd}(p)$. However, if we only consider replacing a node in the set S_{cand}, then we only need to get the CPM entries $P[i, n, o]$ for nodes $n \in S_{cand}$. In this case, we find that we only need to get $P[i, p, q]$ for a subset of nodes p in the circuit and $q \in C_{cd}(p)$. We denote the subset as $N(S_{cand})$. To get $N(S_{cand})$, we initialize a queue W as S_{cand}, and for each node $s \in W$, we move s from W to $N(S_{cand})$ and enqueue each node in the disjoint cut of s except all POs into W. We repeat this step until W is empty.

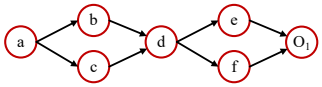


Fig. 6. An example circuit represented in DAG.

Example 2. *Fig. 6 shows an example circuit represented as a DAG with output O_1. Suppose that $S_{cand} = \{a, b\}$. Then W is initialized as $\{a, b\}$. For node a, it is moved from W to $N(S_{cand})$. Since $C_{cd}(a) = \{d\}$, d is enqueued into W. Consequently, $W = \{b, d\}$ and $N(S_{cand}) = \{a\}$. For node b, $C_{cd}(b)$ is still $\{d\}$, which is already in W. Then, we move b from W to $N(S_{cand})$. Consequently, $W = \{d\}$ and $N(S_{cand}) = \{a, b\}$. For node d, $C_{cd}(d) = \{O_1\}$, which is a PO. Then, we move d from W to $N(S_{cand})$. Consequently, $W = \emptyset$ and $N(S_{cand}) = \{a, b, d\}$. We only need to calculate the Boolean differences $P[i, p, q]$ for all $p \in N(S_{cand}) = \{a, b, d\}$ and $q \in C_{cd}(p)$, and other Boolean differences $P[i, n, t]$ with $n \in \{c, e, f\}$ and $t \in C_{cd}(n)$ are not needed. Indeed, this is true, since we only need to get the CPM entries $P[i, a, O_1]$ and $P[i, b, O_1]$, and they can be calculated as follows:*

$$P[i, a, O_1] = P[i, d, O_1] \wedge P[i, a, d],$$
$$P[i, b, O_1] = P[i, d, O_1] \wedge P[i, b, d].$$

In the procedure of calculating the errors of LACs, *i.e.*, step three, some candidate LACs can also be ignored since we only need to consider the LACs whose target nodes are in S_{cand}.

If a node is replaced after applying a LAC, the node and its *maximum fanout free cone (MFFC)* [21] cannot be replaced anymore. Thus, we should remove them from S_{cand}. For example, if node d is replaced in Fig. 6, then node d and its MFFC (nodes a, b, c) are all removed from this circuit, and we should remove them from S_{cand} if S_{cand} contains them.

D. Self-Adaption for the Second Phase

This section proposes two self-adaption techniques for the dual-phase flow to ensure good performance for different circuits: 1) tuning the parameters to improve the efficiency based on a runtime estimation model and 2) adaptively stopping phase two to begin a new dual-phase iteration.

According to Section III-A, the process is initialized with a given size of S_{cand} (*i.e.*, M) and an iteration limit (*i.e.*, N) of the second phase. Denote the number of actual iterations in the second phase as N_r. We have $N_r \leq N$, since S_{cand} may be empty, or the circuit error may exceed the upper bound before the N-th round in the second phase. We also have $N < M$ as

shown in Section III-A. We denote the average runtime of one iteration of phase two as T_{inc} and the runtime of phase one as T_{com}. We have $T_{inc} = f(M)T_{com}$, where $f(M) \leq 1$ is the ratio between T_{inc} and T_{com}. $f(M)$ is an increasing function of M because a larger candidate set S_{cand} leads to more CPM entries and candidate LACs that need to be evaluated in each iteration of phase two. Given that one LAC is applied in the first phase of a dual-phase iteration, (N_r+1) LACs are applied in a dual-phase iteration. The average runtime of applying one LAC in a dual-phase iteration is:

$$T_{avg} = \frac{T_{com} + N_r T_{inc}}{N_r + 1} \approx \left(\frac{1}{N_r + 1} + f(M) \right) T_{com}. \quad (2)$$

We want to minimize T_{avg}. Although decreasing M reduces $f(M)$, it also reduces N_r, since fewer nodes can be replaced in phase two, causing an increase in $\frac{1}{N_r+1}$. Thus, reducing M may not reduce T_{avg}. Additionally, modeling $f(M)$, which depends on the three steps in the error analysis, is complex. Thus, it is hard to solve the best parameters from Eq. (2).

In this work, we choose to focus on the time-dominating step in the *last* dual phase to tune the parameters to reduce T_{avg}. Denote the average runtime of step i ($1 \leq i \leq 3$) to apply a LAC in a dual-phase iteration (*resp.*, the first phase) as T_i (*resp.*, $T_{com,i}$). Step i dominates the runtime if it takes more than half of the total runtime.

We first consider when step 1 dominates the runtime of the last dual phase. To optimize the parameters, we build a rough runtime model. When updating disjoint cut in step 1, we only need to consider the nodes in the set S_v of nodes that violate the CPC. Let Y be the total number of nodes in the circuit and $\overline{|S_v|}$ be the average size of S_v in each iteration of phase two. Then, $T_1 = \frac{1}{N_r+1}(T_{com,1} + N_r \frac{\overline{|S_v|}}{Y} T_{com,1}) \approx \frac{T_{com,1}}{N_r+1} + \frac{\overline{|S_v|} T_{com,1}}{Y}$. The second term of T_1, which corresponds to phase two, is independent of M. This means that increasing M does not increase the workload of step 1 in phase two. However, it allows the consideration of more nodes (*i.e.*, N_r increases). Thus, we can increase M by a factor R_{inc} to decrease the first term of T_1 without affecting the second term, resulting in a lower T_1. Similar analysis can be done when step 2 or 3 dominates the runtime. It shows that we can reduce the runtime of step 2 by tuning M. We can find the optimal M by the information from the last dual phase. If step 3 dominates, we reduce the number of LACs for each target node.

The second self-adaption technique is to adaptively stop phase two according to the error increase. Fig. 4 shows that the remaining nodes in S_{cand} are not always promising to be replaced after some LACs are applied. If we keep S_{cand} until we finally select N LACs, some inferior LACs may be applied to the circuit, which eventually degrades the circuit quality. Thus, we need to stop phase two after some iterations to start a new dual-phase round. We decide when to start a new round based on the error increase of a LAC we select.

Denote the relative error increase of a LAC as e_r. It equals $\frac{E_{inc}}{E_0}$, where E_{inc} is the error increase caused by applying the LAC, and E_0 is the error of the approximate circuit at the beginning of this dual-phase iteration. Let E be the error of the current approximate circuit. We introduce a threshold e_t, a relaxed bound ratio b_r, and a strict bound ratio b_s, where $0 < b_r < b_s < 1$. Given an error upper bound E_b, the flow is divided into the following three parts: 1) If $0 \leq E \leq b_r E_b$, *i.e.*, E is far away from the upper bound, then the relative error increase e_r is unconstrained; 2) If $b_r E_b < E \leq b_s E_b$, then we apply a loose constraint that if the relative error increase e_r of one LAC exceeds e_t, the second phase stops; 3) If $b_s E_b < E \leq E_b$, then we apply a strict constraint that if the sum of the relative error increases in this dual phase exceeds the threshold e_t, the second phase stops.

IV. Experimental Results

This section presents the experimental results. All experiments were carried out on a server with an AMD Ryzen Threadripper PRO 5995WX CPU with 64 cores and 512 GB memory running Ubuntu 22.04. The proposed method can handle different input distributions and different error metrics. In our experiments, all inputs are uniformly distributed. We use the logic synthesis tool ABC [22] and a proprietary standard cell library for technology mapping. Table I lists the benchmark circuits used in our experiments along with their functions, numbers of PIs/POs, node numbers (#Nd) in their AND-inverter graph representations, areas, and delays. The circuits include some ISCAS circuits [23] and some EPFL arithmetic circuits [24]. Additionally, to enhance the variety of benchmarks, we add some other arithmetic circuits. Based on the node numbers of these circuits, they are divided into two groups: a group of small circuits with less than 4000 nodes and a group of large circuits with at least 4000 nodes. The initial number of the candidate nodes M is set as 60 for the small circuits and 150 for the large circuits, and N is initialized as $\frac{M}{3}$. The parameters R_{inc}, b_r, b_s, and e_t used in self-adaption are set as 0.25, 0.025, 0.25, and 0.5, respectively. The number of input patterns for Monte Carlo simulation is 100,000.

TABLE I
Benchmark circuit information.

Circuit	#I/O	Function	#Nd	Area(μm^2)	Delay(ns)
c880	60/26	8-bit ALU	329	69.71	1.8
c1908	33/25	16-bit detector	354	83.00	2.7
c3540	50/22	8-bit ALU	920	190.95	2.9
sm9×8	17/17	9bit×8bit signed multiplier	663	110.34	3.3
sm18×14	32/32	18bit×14bit signed multiplier	2370	362.23	5.6
butterfly	100/72	Radix-2 butterfly	8410	1565.91	8.3
vecmul8	256/35	8-dimensional vector multiplier	16517	3177.63	8.7
mult16	32/32	16-bit unsigned multiplier	3039	527.03	7.1
adder	256/129	128-bit adder	1654	212.56	26.2
sqrt	128/64	128-bit square root unit	22793	4597.77	505.5
sin	24/25	24-bit sin unit	6981	1417.17	18.4
square	64/128	64-bit square unit	25772	4396.47	276.5
log2	32/32	32-bit log2 unit	39022	6560.67	406.1

To evaluate the synthesis quality of the approximate circuit, we use *area-delay product (ADP) ratio*, which is the ADP of the approximate circuit over that of the original circuit. We also report the runtime of different methods. We compare our method with two state-of-the-art methods: 1) VECBEE [19], which applies batch error estimation method based on CPM

to accelerate the process, and 2) AccALS [14], which is based on another efficient error estimation method SEALS [20] and speeds up the process by applying multiple LACs in each iteration. Additionally, we re-implement VECBEE using disjoint cut, which accelerates the process of obtaining CPM. Thus, it is an enhanced version of VECBEE. We also consider two versions of our method, the one without using self-adaption, denoted as DP, and the one with self-adaption, denoted as DP-SA. Given a circuit with K POs, we define a reference error R as $2^{\frac{K}{3}}$. The MED (*resp.*, MSE) thresholds are set as $\{0.5R, R, 2R\}$ (*resp.*, $\{0.5R^2, R^2, 2R^2\}$). The ER thresholds are set as $\{0.1\%, 1\%, 2\%\}$. For small circuits, we consider SASIMI LAC [13] and conduct experiments with three different error thresholds, averaging the results. For large circuits, due to long runtime, we consider only constant LAC (*i.e.*, replacing a node by constant 0 or 1) and conduct one experiment per circuit with the median threshold. Furthermore, for *sqrt* and *log2*, the thresholds are adjusted to smaller values since the baseline method requires an extremely long runtime.

A. Comparison with VECBEE

This section compares the proposed method with the enhanced version of VECBEE under the MSE constraint. Since both methods support multi-threading for error estimation, all experiments in this section were conducted using 16 threads.

TABLE II
COMPARISON OF VECBEE ($l = \infty$), VECBEE ($l = 1$), DP, AND DP-SA ON SMALL AND LARGE CIRCUITS UNDER MSE CONSTRAINT.

Circuit	ADP ratio				Runtime/s			
	$l = \infty$	$l = 1$	DP	DP-SA	$l = \infty$	$l = 1$	DP	DP-SA
c880	**90.5%**	**90.5%**	**90.5%**	**90.5%**	10.1	10.1	4.8	**4.0**
c1908	90.4%	90.4%	**89.9%**	90.3%	15.0	14.5	**4.7**	4.8
c3540	**93.8%**	**93.8%**	94.4%	94.5%	47.2	45.8	**13.4**	13.8
sm9×8	71.7%	72.1%	72.3%	**67.8%**	34.9	38.7	**9.5**	10.7
sm18×14	65.4%	65.7%	66.5%	**64.4%**	910.8	677.1	162.2	**97.8**
mult16	**74.7%**	77.7%	76.8%	75.2%	3160	1436	453.0	**309.5**
adder	44.3%	44.3%	43.8%	**43.7%**	624.5	623.9	94.0	**91.1**
Avg	75.8%	76.3%	76.3%	**75.2%**	686.1	406.6	105.9	**76.0**
sin	**11.1%**	99.7%	11.8%	11.5%	6900	23.0	**450.4**	525.6
square	**59.7%**	65.8%	61.3%	61.1%	130919	91637	**4218**	4967
sqrt	96.8%	**74.6%**	83.3%	83.3%	60187	33271	4067	**4042**
log2	79.4%	87.3%	75.1%	**74.5%**	85420	28173	3652	**3525**
butterfly	**68.5%**	94.1%	71.1%	70.3%	15684	3325	**2234**	2748
vecmul8	68.3%	72.0%	69.2%	**68.0%**	39808	20176	**911.5**	4548
Avg	64.0%	82.2%	62.0%	**61.4%**	56486	29434	**2589**	3393

1) Performance on Small Circuits: The error estimation accuracy of VECBEE [19] is configurable by a parameter, depth limit l. For example, if $l = \infty$, it is fully accurate. If $l = 1$, then the cut in Eq. (1) is replaced by its direct fanout, which is the most efficient but sometimes inaccurate. We compare two versions of our methods, DP and DP-SA, with two versions of VECBEE with $l = \infty$ and $l = 1$ on the small circuits, as shown in the upper half of Table II. For each circuit, we highlight the best result among all methods in **bold**. The minimum average ADP ratio and runtime are both given by our method DP-SA. Compared with DP, the self-adaption techniques can both speed up on small circuits ($1.4\times$ faster) and improve the circuit quality (1.1% lower ADP). Compared to VECBEE, DP-SA is $5.4\times$ faster than the efficient version of VECBEE and $9.0\times$ faster than the accurate version of VECBEE with a slightly better ADP.

2) Performance on Large Circuits: To show the scalability of our method, we compare it with both versions of VECBEE on large circuits. The results are shown in the lower half of Table II. Compared with DP, self-adaption improves ADP on all circuits except *sqrt*, while is $1.3\times$ slower. This is because self-adaption is more sensitive to a large error increase, and it sometimes stops the second phase more frequently to avoid applying an inferior LAC. Compared to VECBEE with $l = \infty$, the efficient version of VECBEE with $l = 1$ only speeds up by $1.9\times$ with 18.2% higher ADP ratio, while our method DP achieves a speed-up of $21.8\times$ without any quality loss.

B. Comparison with AccALS

This section compares the proposed method, DP-SA, with AccALS [14]. To show the robustness of our method, we consider two other error metrics, ER and MED. Since AccALS does not support multi-threading, all experiments in this section were conducted using a single thread. The results are shown in Table III. Under the ER error metric, both methods have advantages in different circuits. However, under the MED error metric, DP-SA is slower only in *c1908* and *log2*, but it achieves 1.8% and 10.7% lower ADP, respectively, for these two circuits. This is because when evaluating MED, accurate error estimation is more important compared to ER. In most iterations, AccALS finds that after applying multiple LACs, there is a large deviation between the real MED and the estimated MED. Thus, it only selects one LAC in this round, which is essentially a SEALS flow [20]. On average, our method is $5.2\times$/$2.1\times$ faster under the ER/MED constraint, while the average ADP of the circuits synthesized by our method is 2.5%/2.9% smaller.

TABLE III
COMPARISON OF ACCALS AND DP-SA UNDER ER/MED CONSTRAINTS.

Circuit	ER				MED			
	ADP ratio		Runtime/s		ADP ratio		Runtime/s	
	AccALS	DP-SA	AccALS	DP-SA	AccALS	DP-SA	AccALS	DP-SA
c880	89.9%	**89.0%**	**7.2**	9.4	**90.9%**	91.0%	26.8	**21.1**
c1908	94.9%	**84.1%**	**4.5**	18.7	92.9%	**91.1%**	**24.1**	25.0
c3540	**96.6%**	97.5%	**11.4**	27.3	95.6%	**95.1%**	85.2	**41.6**
sm9×8	**98.3%**	**98.3%**	13.3	**5.6**	56.5%	**55.6%**	390.4	**21.2**
sm18×14	**94.5%**	**94.5%**	127.7	**30.2**	76.7%	**64.5%**	1981	**142.6**
mult16	**87.3%**	**87.3%**	22.2	**8.0**	**74.2%**	74.8%	2170	**995.5**
adder	**99.1%**	**99.1%**	55.7	**53.8**	59.2%	**58.2%**	180.5	**116.0**
sin	78.6%	**74.8%**	**194.0**	972.4	18.3%	**18.0%**	13336	**1824**
square	99.2%	**94.4%**	**715.3**	3906	**92.1%**	92.4%	29570	**9957**
sqrt	84.3%	**75.7%**	108182	**13593**	**66.5%**	72.2%	428222	**215869**
log2	72.1%	**67.8%**	42477	**9795**	96.4%	**85.7%**	**6492**	10252
butterfly	96.4%	**96.3%**	602.1	**253.4**	**72.8%**	73.9%	31377	**4530**
vecmul8	**99.1%**	99.2%	**681.5**	871.0	88.4%	**69.8%**	7808	**2929**
Avg	91.6%	**89.1%**	11776	**2272**	75.4%	**72.5%**	40128	**18979**

V. CONCLUSION

In this work, we propose a novel ALS method based on a dual-phase iterative framework. Our flow performs a round of comprehensive analysis in the first phase and multiple rounds of incremental analysis in the second phase, in which we skip unnecessary computation, thus effectively accelerating the ALS flow. It outperforms the state-of-the-art methods in runtime together with an improvement in circuit quality.

REFERENCES

[1] M. M. Waldrop, "The chips are down for Moore's law," *Nature News*, vol. 530, no. 7589, p. 144, 2016.

[2] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *European Test Symposium*, 2013, pp. 1–6.

[3] Q. Xu, T. Mytkowicz, and N. S. Kim, "Approximate computing: A survey," *IEEE Design & Test*, vol. 33, no. 1, pp. 8–22, 2015.

[4] N. Zhu, W. L. Goh, G. Wang, and K. S. Yeo, "Enhanced low-power high-speed adder for error-tolerant application," in *International SoC Design Conference*, 2010, pp. 323–327.

[5] W. Liu, L. Qian, C. Wang, H. Jiang, J. Han, and F. Lombardi, "Design of approximate radix-4 Booth multipliers for error-tolerant computing," *IEEE Transactions on Computers*, vol. 66, no. 8, pp. 1435–1441, 2017.

[6] A. Agrawal, J. Choi, K. Gopalakrishnan, S. Gupta, R. Nair, J. Oh, D. A. Prener, S. Shukla, V. Srinivasan, and Z. Sura, "Approximate computing: Challenges and opportunities," in *International Conference on Rebooting Computing*, 2016, pp. 1–8.

[7] R. Ye, T. Wang, F. Yuan, R. Kumar, and Q. Xu, "On reconfiguration-oriented approximate adder design and its application," in *International Conference on Computer-Aided Design*, 2013, pp. 48–54.

[8] D. Shin and S. K. Gupta, "Approximate logic synthesis for error tolerant applications," in *Design, Automation & Test in Europe*, 2010, pp. 957–960.

[9] A. Chandrasekharan, M. Soeken, D. Große, and R. Drechsler, "Approximation-aware rewriting of AIGs for error tolerant applications," in *International Conference on Computer-Aided Design*, 2016, pp. 1–8.

[10] D. Shin and S. K. Gupta, "A new circuit simplification method for error tolerant applications," in *Design, Automation & Test in Europe*, 2011, pp. 1–6.

[11] C. Meng, W. Qian, and A. Mishchenko, "ALSRAC: Approximate logic synthesis by resubstitution with approximate care set," in *Design Automation Conference*, 2020, pp. 1–6.

[12] J. Ma, S. Hashemi, and S. Reda, "Approximate logic synthesis using Boolean matrix factorization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 1, pp. 15–28, 2021.

[13] S. Venkataramani, K. Roy, and A. Raghunathan, "Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits," in *Design, Automation & Test in Europe*, 2013, pp. 1367–1372.

[14] X. Wang, S. Tao, J. Zhu, Y. Shi, and W. Qian, "AccALS: Accelerating approximate logic synthesis by selection of multiple local approximate changes," in *Design Automation Conference*, 2023, pp. 1–6.

[15] J. Echavarria, S. Wildermann, and J. Teich, "Approximate logic synthesis of very large Boolean networks," in *Design, Automation & Test in Europe*, 2021, pp. 1552–1557.

[16] M. Barbareschi, S. Barone, N. Mazzocca, and A. Moriconi, "A catalog-based AIG-rewriting approach to the design of approximate components," *IEEE Transactions on Emerging Topics in Computing*, vol. 11, no. 1, pp. 70–81, 2022.

[17] L. Witschen, T. Wiersema, M. Artmann, and M. Platzner, "MUSCAT: MUS-based circuit approximation technique," in *Design, Automation & Test in Europe*, 2022, pp. 172–177.

[18] C.-T. Lee, Y.-T. Li, Y.-C. Chen, and C.-Y. Wang, "Approximate logic synthesis by genetic algorithm with an error rate guarantee," in *Asia and South Pacific Design Automation Conference*, 2023, pp. 146–151.

[19] S. Su, C. Meng, F. Yang, X. Shen, L. Ni, W. Wu, Z. Wu, J. Zhao, and W. Qian, "VECBEE: A versatile efficiency–accuracy configurable batch error estimation method for greedy approximate logic synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 11, pp. 5085–5099, 2022.

[20] C. Meng, X. Wang, J. Sun, S. Tao, W. Wu, Z. Wu, L. Ni, X. Shen, J. Zhao, and W. Qian, "SEALS: Sensitivity-driven efficient approximate logic synthesis," in *Design Automation Conference*, 2022, pp. 439–444.

[21] K. Juretus and I. Savidis, "Increasing the SAT attack resiliency of in-cone logic locking," in *International Symposium on Circuits and Systems*, 2019, pp. 1–5.

[22] A. Mishchenko *et al.*, "ABC: A system for sequential synthesis and verification," http://people.eecs.berkeley.edu/~alanmi/abc/, 2024.

[23] M. C. Hansen, H. Yalcin, and J. P. Hayes, "Unveiling the ISCAS-85 benchmarks: A case study in reverse engineering," *IEEE Design & Test of Computers*, vol. 16, no. 3, pp. 72–80, 1999.

[24] EPFL Integrated Systems Lab, "The EPFL combinational benchmark suite," https://github.com/lsils/benchmarks, 2024.