Automated Verifiability-Driven Design of Approximate Circuits: Exploiting Error Analysis

Zdenek Vasicek, Vojtech Mrazek, Lukas Sekanina
Brno University of Technology, Faculty of Information Technology, Brno, Czech Republic
Email: {vasicek, mrazek, sekanina}@fit.vutbr.cz

***Abstract*—A fundamental assumption for search-based circuit approximation methods is the ability to massively and efficiently traverse the search space and evaluate candidate solutions. For complex approximate circuits (adders and multipliers), common error metrics, and error analysis approaches (SAT solving, BDD analysis), we perform a detailed analysis to understand the behavior of the error analysis methods under constrained resources, such as limited execution time. In addition, we show that when evaluating the error of a candidate approximate circuit, it is highly beneficial to reuse knowledge obtained during the evaluation of previous circuit instances to reduce the total design time. When an adaptive search strategy that drives the search towards promptly verifiable approximate circuits is employed, the method can discover circuits that exhibit better trade-offs between error and desired parameters (such as area) than the same method with unconstrained verification resources and within the same overall time budget. For 16-bit and 20-bit approximate multipliers, it was possible to achieve a 75% reduction in area when compared with the baseline method.**

I. Introduction

Approximate circuits showing desired trade-offs between the error (in other words, the output quality) and electrical parameters (such as power consumption, area, and delay) are crucial components of energy-efficient implementations of deep neural networks (DNN) [1]. The design of approximate circuits can be based on selecting a suitable exact initial implementation, intelligent generating its modifications, and checking whether they satisfy the specification [2], [3]. With such a *search-based design method*, many circuits are naturally obtained; however, those showing the best trade-offs (i.e., occupying the Pareto front) are the most important. While the electrical parameters can be quickly estimated with acceptable precision during the search, evaluating the error is a challenging problem, especially for complex arithmetic circuits. Hence, methods of formal verification such as SAT solving and properties analysis using *Binary Decision Diagrams* (BDDs) have been employed to determine the error [4].

In this paper, we perform a detailed analysis of error computing methods used in search-based circuit approximation algorithms. The basic assumption of the analysis is that (i) the exact error analysis is always conducted (i.e., we exclude error estimation methods based on simulation or probability analysis [1]), (ii) the error computing method is called many times (thousands to millions times [2]) during the design and its outputs can be reused, (iii) the error computing method can be terminated under some conditions to save resources, and (iv) the search algorithm has a fixed time budget.

Accelerating the error analysis is critical because more candidate approximate designs can then be explored within the available time, and circuits showing better trade-offs can thus be reached. Furthermore, quickly evaluating and eliminating candidate approximate circuits that do not satisfy the constraint further accelerates and improves the design process.

For typical approximate arithmetic circuits (20-bit adders and 8-bit multipliers), common error metrics ($WCAE$, MAE) and error analysis approaches (SAT solving, BDD analysis), we aim to determine the best-performing strategy for incorporating constraints into the error analysis method to minimize the overall circuit approximation time. Hence, we developed an experimental platform to analyze the role of verification engines in search-based circuit approximation methods. During the experimental evaluation, we revealed some useful knowledge. For example, we learned that (a) proving the error constraint can be difficult for one method while being easy for the other, and vice versa; (b) caching intermediate verification results significantly improves the performance; (c) despite introducing constraints for verification engine, well-performing circuits are not lost.

After a brief survey of the state-of-the-art in Section II, these results are presented in Section III and IV. The obtained knowledge is then utilized in an improved version of the approximation method and evaluated on complex approximate multipliers in Section V. Conclusions are given in Section VI.

II. Automated design of approximate circuits

Let $f : \mathbb{B}^n \rightarrow \mathbb{B}^m$ be an n-input m-output Boolean function that describes correct functionality (specification) and $\hat{f} : \mathbb{B}^n \rightarrow \mathbb{B}^m$ be an approximation of it, both implemented by two circuits, namely F and $\hat{\text{F}}$. The *worst-case arithmetic error* ($WCAE$) is defined as

$$e_{wcae}(f, \hat{f}) = \max_{\forall x \in \mathbb{B}^n} |\,\text{nat}(f(x)) - \text{nat}(\hat{f}(x))| \qquad (1)$$

where $\text{nat}(x)$ represents a function $\text{nat} : \mathbb{B}^m \rightarrow \mathbb{Z}$ returning a decimal value of the m-bit binary vector x.

The *mean absolute error* (MAE) is defined as the sum of absolute differences in magnitude between the original and approximate circuits averaged over all inputs:

$$e_{mae}(f, \hat{f}) = \frac{1}{2^n} \sum_{\forall x \in \mathbb{B}^n} |\,\text{nat}(f(x)) - \text{nat}(\hat{f}(x))| \qquad (2)$$

After introducing these basic error metrics, we will briefly characterize relevant circuit approximation methods.

979-8-3503-4859-0/DATE24/© 2024 EDAA

A. Circuit Approximation Methods

Approximate implementations of digital circuits are often obtained by the so-called *functional approximation* [3]. This method starts with an original (exact) circuit and tries to modify its logic behavior (and the subsequent implementation) in such a way that the best possible trade-offs between the quality of output (the error) and electrical characteristics (such as power consumption, area, and delay) are obtained.

The basic algorithmic approximation techniques are pruning (i.e., removing some parts of the circuit), component replacement (i.e., complex subcircuits are replaced with simpler subcircuits), and approximate re-synthesis [3]. The automated approximation methods select either randomly or heuristically which parts of the circuit have to be removed, re-connected or replaced. An automated circuit approximation method can be seen as a multi-objective search process in which a circuit satisfying user-defined constraints is sought within the space of all possible implementations [2]. The search is driven by an objective function g. For example, if the goal is to minimize circuit cost and ensure that the chosen error metric $\mathcal{E}$ (e.g., e_{wcae} or e_{mae}) is less than a threshold τ then:

$$g(\hat{\mathrm{F}}, \tau) = \begin{cases} cost(\hat{\mathrm{F}}) & \text{if } \mathcal{E}(\mathrm{F}, \hat{\mathrm{F}}) \leq \tau \\ \infty & \text{otherwise.} \end{cases} \quad (3)$$

As in [5], it can safely be assumed that the cost can be estimated, for example, as the sum of weighted areas of the gates used in the circuit. The search can be conducted by various algorithms, for example, by *Cartesian Genetic Programming* (CGP) [5] or Jump Search [6].

B. Error Calculation

In addition to simulation techniques and probabilistic methods, formal verification techniques have been adopted to evaluate the error of approximate circuits [4]. They are often based on *equivalence checking*, i.e., checking whether a mathematical model of a circuit under design meets a given specification. Two main approaches have been developed in this direction – techniques based on *Reduced Ordered Binary Decision Diagrams* (BDDs) [7] and Satisfiability (SAT) solvers [8]. Candidate approximate circuits are checked to be equal up to some bound w.r.t. a chosen error metric.

The main advantage of formal verification is that it can provide the exact error even for large circuits. Moreover, the error is obtained much faster than using simulation in many cases [4]. Several studies have proven that the time needed to determine the error depends on many factors, including the circuit type, error metric, and the actual error magnitude [5].

For example, Ceska et al. [5] showed on complex approximate multipliers that the search algorithm can be forced to generate candidate approximate circuits that can quickly be verified. They constrained the verification time for each candidate circuit to generate and evaluate more circuits within a fixed design time budget. Better trade-offs were then reached by the search method utilizing constrained verification resources compared with the same method utilizing unconstrained resources. The second approach spent most of the time verifying some hard instances ignoring thus more promising cases. A method determining reasonable constraints prior to search space exploration and independent of employed approximation methods was presented in [9].

III. Analysis of Error Computation Cost

Various algorithms have been proposed to compute error metrics [4], [10]–[12]. Determining which one will perform best when used in the automated circuit approximation loop is not trivial. The reasons are twofold. First, the performance of the checking algorithms is typically evaluated under different conditions (and on significantly fewer instances) than the search-based methods assume. Second, the algorithms can behave differently contrasted to their common utilization, e.g., due to the presence of cache and similarity between generated candidate design points.

This section aims to experimentally analyze the computational complexity of the most frequent error metrics when used in automated circuit approximation. This analysis focuses on arithmetic error, which is relevant to the arithmetic blocks such as multipliers and adders.

There are two basic possibilities for implementing the condition $\mathcal{E}(\mathrm{F}, \hat{\mathrm{F}}) \leq \tau$ in Eq. 3. First, the value of the error metric, i.e. $\mathcal{E}(\mathrm{F}, \hat{\mathrm{F}})$, can be computed using a suitable formal method and used to evaluate the condition. Second, we can check that the inequality holds without having to calculate $\mathcal{E}(\mathrm{F}, \hat{\mathrm{F}})$, i.e. by formally proving that the whole condition holds. We will denote the first as *error analysis*, and *error checking* the latter. The first approach requires performing so-called model counting (SAT counting), whose computational complexity is known to be #P-complete. BDDs or #SAT solvers can be used to accomplish this task. BDDs can be used to effectively analyze large circuits (such as 128-bit adders) in a few seconds on a common PC but do not scale well for some circuits such as multipliers. #SAT solvers exhibit poor performance in error analysis even for modest instances like 8-bit multipliers, making them slower than exhaustive simulation methods. Hence, some authors relax the requirement for formal guarantee and use approximate model counting [13]. The second approach (error checking) is less computationally expensive, but still, it is NP-complete.

We demonstrate the behavior of formal methods on two approximate circuit instances common in real applications – 8-bit multipliers and 20-bit adders. The choice of these examples is not arbitrary as we have already collected and evaluated a large number of different design points (8-bit multipliers) and we can evaluate the performance of the formal methods in a reasonable amount of time.

Methodology: We systematically selected approximate circuits from the full version of the EvoApproxLib [14] so that representative, comprehensive and uniformly covered subset of the entire design space is obtained for our experiments (see Fig. 1). Every design point is defined using the *Area-Delay Product* (ADP) and the logarithm of the investigated error metrics. For 8-bit multipliers, the extracted circuits effectively

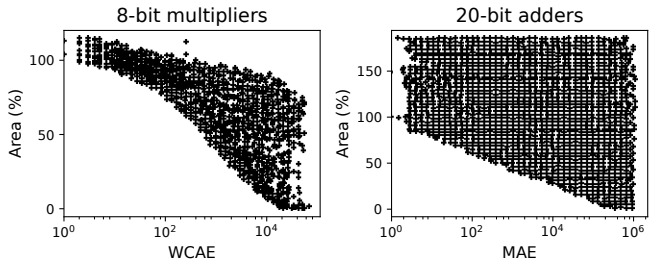

Fig. 1: Properties of 1150 approximate multipliers (left) and 1600 adders (right) considered in our study. The area is given relative to the exact 8-bit Array Multiplier and 20-bit Ripple-Carry adder. Note that circuits with an area above 100% have lower delay compared to those that are more compact.

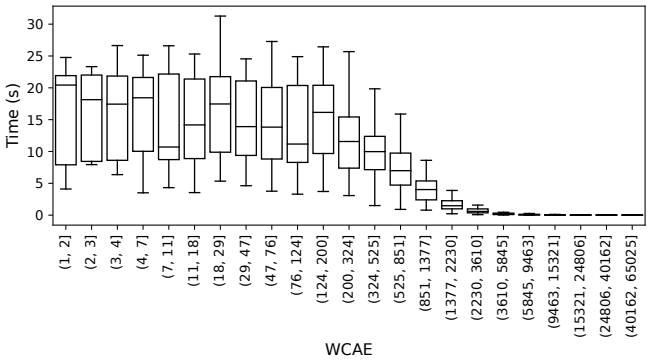

Fig. 2: The time required to calculate $\mathrm{WCAE}(\mathrm{F}, \hat{\mathrm{F}})$ for approximate 8-bit multipliers depending on the actual WCAE value of the multiplier.

span the entire relevant design space, i.e., Pareto-optimal[1] as well as sub-optimal solutions are covered. For each design point, we executed algorithms that are believed to be the most efficient to determine the $WCAE$ and MAE, employing SAT solver (MiniSAT) for the former and BDDs (Buddy) for the latter. The computations involved determining the errors across a specified range of thresholds (i.e., τ values). The experiments were conducted on a machine equipped with 16-core Intel Xeon E5-2670 CPU running at 2.6 GHz with 64 GB RAM. The BDD used a cache for 10 million nodes and variable ordering optimal for adders. The following exact reference circuits denoted as F were chosen: an 8-bit array multiplier for multipliers and a 20-bit ripple-carry adder for adders. It is worth noting that the reference itself influences the runtime of error checking and error analysis [4], but its impact can be omitted in our study.

A. WCAE

The whole condition $WCAE(\mathrm{F}, \hat{\mathrm{F}}) \leq \tau$ as well as the value of $WCAE(\mathrm{F}, \hat{\mathrm{F}})$ itself can be computed formally using a SAT-solver [4]. A form of binary search is usually employed in the latter case.

Fig. 2 shows the runtimes obtained for the error analysis of 8-bit approximate multipliers using Algorithm 5 proposed in [4] which is implemented using incremental SAT. The x-axis is segmented into bins, uniformly distributed across the possible range of $WCAE$ (1 to 65025 for the 8-bit multiplier), and evenly spaced in the logarithmic domain. It would be more precise to present the number of SAT conflicts to show the computational complexity because it is invariant to the system load. However, we present the runtime to keep the results consistent across the entire section.

We can observe a strong dependence of the computation time on the actual value of $WCAE$. When such a method is used in the automated circuit approximation loop, the performance varies depending on the error of the generated candidate solutions. The differences are huge – up to two orders of magnitude. The maximum recorded runtime is approximately 31 s ($WCAE = 27$) which corresponds with 923,527 SAT conflicts and 1,017,723 decisions. On the contrary, 11 ms are required on the average to determine the error for instances with the highest $WCAE$ (i.e. in the range 40162 - 65025). In those cases, less than 628 SAT conflicts and 506 decisions are reported on the average. In summary, the design of multipliers with a low $WCAE$ will be more challenging compared to the design of multipliers having large errors.

Fig. 3 shows the run time of the error checking, i.e., proving that $WCAE$ is greater than a given threshold τ. Because the construction of the miter (for details, refer to [5]) depends on the chosen threshold, results for each threshold must be presented separately. Due to the limited space, eight thresholds linearly sampled in the log space are included in Fig. 3. Two trends can be observed. First, the lower the threshold, the higher the computational complexity of the checking. This corresponds with the findings for $WCAE$ error analysis. Second, the lower the actual value of $WCAE$, the higher the computation complexity. As the current $WCAE$ error value of the checked circuit moves farther away from τ in the direction of lower values, the computational complexity increases. Conversely, detecting threshold violations is feasible within a few milliseconds.

Comparing the runtimes of error analysis with the error checking clearly shows that the error checking is substantially more computationally efficient despite the mentioned dependencies causing high discrepancies between run-times. It is thus more efficient to use error checking to determine the validity of Eq. 3, especially for small τ.

Note that $WCAE$ analysis as well as checking of adders is substantially easier compared to the multipliers. Even the 32-bit instances can be evaluated in tens of milliseconds which aligns with results presented in [4].

B. MAE

The computation of the MAE requires model counting, which, conversely to the previous case, can be performed

[1]By Pareto-optimal solutions, we mean a set of non-dominated solutions showing the best-known trade-offs.

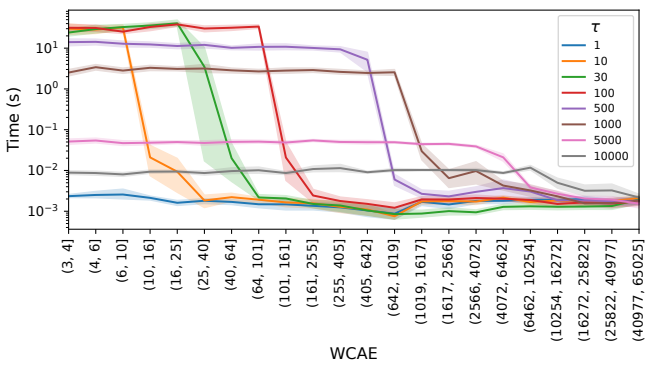

Fig. 3: The time required to prove $WCAE(\mathrm{F}, \hat{\mathrm{F}}) > \tau$ for 8-bit approximate multipliers depending on the actual $WCAE$ value of the multiplier. The maximum, average, and minimum values are reported.

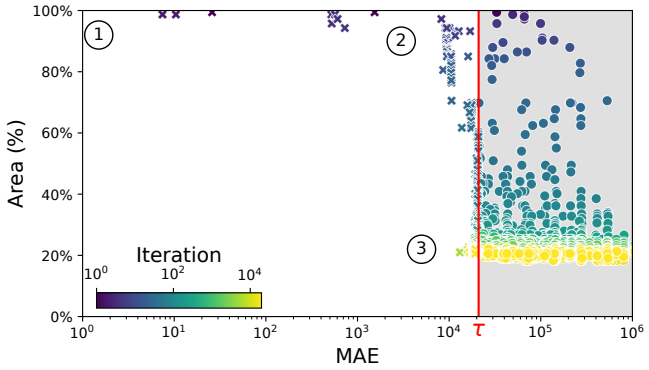

Fig. 5: Candidate solutions explored during the approximation of a 20-bit RCA adder for $\tau = 20972$. Note that 10000 iterations correspond to 450 seconds; the crosses and circles denote the acceptable and unacceptable solutions, respectively.

more efficiently using BDDs due to their significantly lower computational complexity compared to the usage of #SAT solver [4]. As the computation operates over a BDD model representing the miter, it is not possible to evaluate the whole condition in Eq. 3 directly at the level of BDDs. It is thus always necessary to determine $WCAE(\mathrm{F}, \hat{\mathrm{F}})$ and then compare the computed value with τ.

We employed Algorithm 4 from [11] to evaluate the computational cost of MAE analysis. In Fig. 4, we present the overall run-time for 20-bit approximate adders involving the BDD construction and MAE computation. It is worth emphasizing that every adder has been evaluated in an isolated way to avoid discrepancies due to the presence of a cache in the used BDD library. The results show a direct correlation between the computation time and the rising MAE values, with one exception -— the instances exhibiting the highest error do not follow this trend. Interestingly, this trend is completely opposite to the $WCAE$.

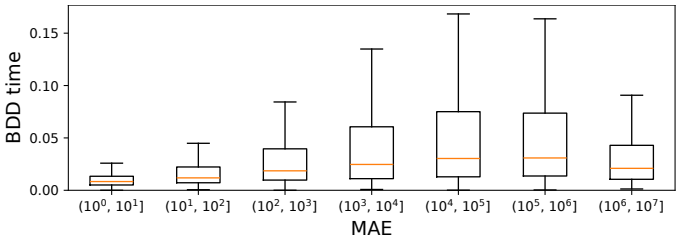

Fig. 4: The time required to determine $MAE(\mathrm{F}, \hat{\mathrm{F}})$ for 20-bit approximate adders depending on the actual MAE value.

IV. Impact of Resource Limit

Section III showed that the time needed to determine the validity of the condition in Eq. 3 depends significantly on the actual error of the circuit we are investigating and also on τ. As τ is typically a constant design parameter during the design process, the approximation method aims to find an approximated circuit whose error is as close as possible to τ and whose other parameters (e.g., area and delay) are minimized.

From this point of view, we can divide the candidate approximate circuits into *acceptable* ones (those that have an error smaller than the threshold) and *unacceptable* ones. Unfortunately, we do not know the real error of a candidate circuit before its evaluation, and therefore, we are unable to selectively choose only acceptable solutions during the search. To minimize the verification time, however, we can constrain some resources. In the case of SAT solving, it can be the maximum allowed number of conflicts (see e.g. [5]); in the case of BDDs it can be the maximum allowed size of the BDD representation (the computation can be terminated during the construction of the BDD, which is the most time- consuming part), or in general we can introduce a limit on the maximum computation time.

Introducing a limit L on verification resources can impact the verification procedure differently. In the case of SAT-based $WCAE$ computation, Fig. 3 suggests that terminating the computation when the maximum allowed number of conflicts (or time) is exceeded will necessarily lead to the loss of otherwise acceptable solutions. This is because it takes much longer to check the condition $WCAE(\mathrm{F}, \hat{\mathrm{F}}) < \tau$. On the contrary, the opposite effect can be expected in the case of BDD-based MAE computation since the computation time increases with the size of the error. It means that it takes much less time to verify that $MAE(\mathrm{F}, \hat{\mathrm{F}}) < \tau$. Introducing a threshold in the case of BDDs can accelerate the calculation because unacceptable candidate solutions will mostly be ignored.

Before discussing the performance of the presented algorithms, it is worth describing a trajectory along which the design method typically moves. Following [11], we consider the search strategy that aims to minimize the area while keeping the MAE below τ. A particular example for $\tau = 20972$ (corresponding to 1% MAE) is shown in Fig. 5. In the first few iterations (around 9 iterations for our example, which corresponds to a few milliseconds), the initial exact circuit (see

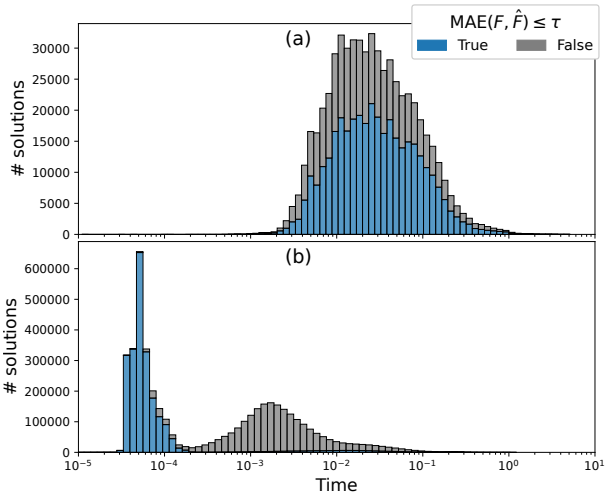

Fig. 6: Distribution of time required to analyse MAE when a) each design point is analysed in isolation, b) the analysis can reuse BDD nodes from the past. All design points generated during the approximation of a 20-bit adder and 15 values of τ are considered.

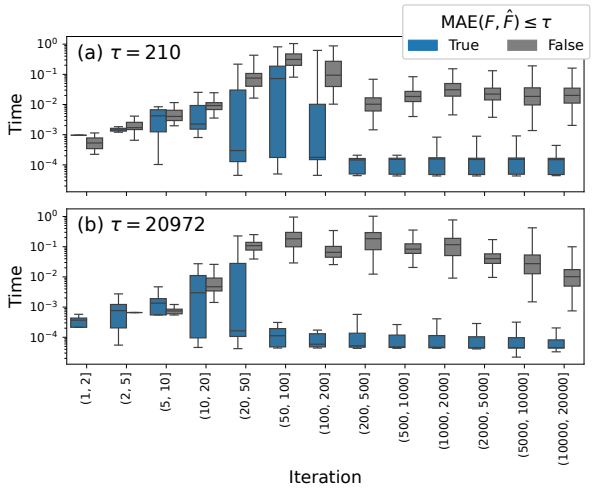

Fig. 7: The time required to analyze MAE of the generated approximate 20-bit adders for small (0.01%) and large (1%) τ during the course of design process.

①) is gradually modified until the error hits τ and reaches the corner depicted ②. During this phase, the area only slightly decreases. In the second phase, the error remains close to τ. The area is gradually reduced until it reaches the solution close to final circuit implementation marked as ③. It takes another 340 iterations (15 seconds) in our example. Then, circuits near ③ are generated for the remaining number of iterations. Note that the maximum allowed number of iterations is intentionally large to demonstrate various effects that will be discussed in the following part.

We must emphasize that most of the time is spent verifying unacceptable solutions concerning the maximum allowed threshold. For example, 21.8% unacceptable solutions were generated within the experiment shown in Fig. 5, which together contribute to 77.8% of the total computation time. This finding is crucial for understanding the computational complexity of the design method in terms of the behavior of formal error checking and analysis. Our findings in Section III reveal that the search-based approximation would highly benefit from SAT-based $WCAE$ error checking. First, unacceptable solutions are verified very quickly. Second, the candidate solutions that are lost due to using the resource constraints may not be important. When the BDD-based MAE computation is employed, the introduced resource constraint improves the performance because the constraint causes the analysis to fail quickly, mostly for unacceptable solutions.

Because many structurally similar candidate solutions are generated during the course of approximation, another property – the similarity of the generated candidate solutions – can be exploited to reduce the execution time. While quantifying the degree of circuit similarity is inherently non-trivial, we can use a proxy for that purpose. The BDD library inherently contains a cache that optimizes memory retrieval and speeds up subsequent computations, enhancing overall efficiency. As the cache is shared among computations, its presence facilitates reduced computation time when similar Boolean functions are present during the computations.

This effect is illustrated in Fig. 6, showing the distribution of time needed to analyze whether the constraint of Eq. 3 is satisfied during the automated approximation of a 20-bit adder. Two approaches are compared: (1) each candidate solution is evaluated in an isolated way (history doesn't impact the execution time) and (2) leveraging the cached information is enabled. In case (1), there is no significant difference between acceptable and unacceptable solution distributions. In case (2), however, not only the mean value decreases, but also the acceptable and unacceptable solutions are clearly separated. This feature can be exploited to accelerate the design exploration process significantly. Setting a suitable resource limit will help to reject the majority of the unacceptable solutions. For $L = 1$ ms, for example, 72.6% of unacceptable solutions can be rejected. The limit leads to total savings of 2,111 s, corresponding to the 7-fold speedup. Unfortunately, we do not know the optimal value of L before the search is started. Hence, it is thus useful to adapt L during the search dynamically using collected statistics.

The need for an adaptive strategy to determine L also arises from the fact that smaller and smaller instances need to be verified as the search progresses. This effect is visible in Fig. 7, showing the time needed to analyze the validity of the constraint (Eq. 3) during the course of the design process. For $\tau = 20972$, for example, the time needed to calculate MAE remains stable after 50 iterations and is significantly lower than in the previous iterations. In addition, the gap between the time needed to analyze unacceptable and acceptable solutions is widening, emphasizing the difference in computational demands for these distinct categories.

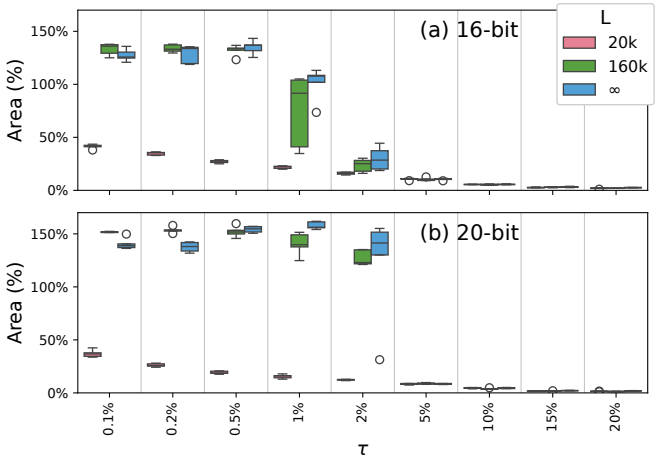

Fig. 8: Parameters of approximate 16-bit and 20-bit multipliers for target $WCAE$ threshold τ designed by the search-based method equipped with SAT-based error checking with given limit L to the maximum allowed number of SAT conflicts.

V. Automated Approximation with Constrained Verification Resources

To demonstrate the impact of constraining the verification engine on the performance of the search-based circuit approximation method, we selected two more challenging design cases – 16-bit and 20-bit multipliers. CGP seeded with array multiplier is used to minimize the area while keeping $WCAE$ below a given τ (9 values considered) as in Eq. 3. This requirement on $WCAE$ is checked by a SAT solver whose runtime is constrained by the maximum allowed number of the SAT conflicts L, $L = \{20 \cdot 10^3, 160 \cdot 10^3, \infty\}$. The limit allows us to terminate the SAT solver early in the case of hard instances. For example, in the case of 16-bit multipliers, $L = 20 \cdot 10^3$ and $160 \cdot 10^3$, the solver is terminated after approx. 3 s and 120 s on average, respectively. We executed 30 independent CGP runs for each circuit and τ; a single run took up to 2 hours. Note that L is constant across thresholds to simplify the situation. The remaining setup is identical with [5].

Fig. 8 shows the box plots for the area and every L and τ. The area after synthesis carried out with Synopsys Design compiler (45nm PDK library and compile_ultra command) normalized to the area on a chip of the initial exact circuit is provided. As evident from the figure, the most aggressive resource limit $L = 20 \cdot 10^3$ allowed to improve the area-error trade-offs significantly. The result aligns with the observations presented within Section III-A, indicating that the lower the τ, the higher the computational cost. The introduction of the limit has proven to be highly beneficial, as it allows for the exploration of a greater number of solutions within the same time budget, thus approaching the Pareto optimum solutions more closely. Furthermore, this illustrates that a high-quality trade-off can be attained, even if certain acceptable solutions are discarded during the design space exploration.

VI. Conclusions

This paper analyzed the impact of a resource limit introduced into the verification process within an automated search-based approximation of digital circuits. Two representative case studies were chosen to highlight various effects of constrained verification on two typically used arithmetic error metrics and formal models – $WCAE$ (computed using a SAT solver) and MAE (computed using BDDs).

We observed contrasting challenges in error verification between $WCAE$ and MAE, where proving the acceptability of error for one method was found to be difficult while being easy for the other, and vice versa. Introducing a resource limit into verification raised concerns about potentially losing valuable candidate solutions; however, experimental evaluation demonstrated that such concerns were unfounded. Furthermore, our investigation revealed the exploitable similarity among candidate solutions during the verification process. While this similarity is leveraged implicitly by BDDs, we have not yet utilized this similarity in the context of SAT. We intend to explore its potential applications in our future work.

Acknowledgements

This work was supported by the Czech Science Foundation project under number 24-10990S.

References

[1] A. Bosio, D. Menard, and O. Sentieys, *Approximate Computing Techniques: From Component- to Application-Level*. Springer Cham, 2022.

[2] L. Sekanina, Z. Vasicek, and V. Mrazek, *Automated Search-Based Functional Approximation for Digital Circuits*. Springer International Publishing, 2019, pp. 175–203.

[3] I. Scarabottolo, G. Ansaloni *et al.*, "Approximate logic synthesis: A survey," *Proceedings of the IEEE*, vol. 108, no. 12, pp. 2195–2213, 2020.

[4] Z. Vasicek, "Formal methods for exact analysis of approximate circuits," *IEEE Access*, vol. 7, no. 1, pp. 177 309–177 331, 2019.

[5] M. Ceska, J. Matyas *et al.*, "Approximating complex arithmetic circuits with formal error guarantees: 32-bit multipliers accomplished," in *Proc. of 36th IEEE/ACM Int. Conf. On Computer Aided Design*. IEEE, 2017, pp. 416–423.

[6] L. Witschen, H. G. Mohammadi *et al.*, "Jump search: A fast technique for the synthesis of approximate circuits," in *Proceedings of the 2019 on Great Lakes Symposium on VLSI, GLSVLSI*. ACM, 2019, pp. 153–158.

[7] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Computers*, vol. 35, no. 8, pp. 677–691, 1986.

[8] A. Biere, M. Heule *et al.*, Eds., *Handbook of Satisfiability - Second Edition*, ser. Frontiers in Artificial Intelligence and Applications. IOS Press, 2021, vol. 336.

[9] L. Witschen, T. Wiersema *et al.*, "Search space characterization for approximate logic synthesis," in *DAC '22: 59th ACM/IEEE Design Automation Conference*, R. Oshana, Ed. ACM, 2022, pp. 433–438.

[10] M. Soeken, D. Große *et al.*, "BDD minimization for approximate computing," in *ASP-DAC '16*, 2016, pp. 474–479.

[11] V. Mrazek, "Optimization of BDD-based approximation error metrics calculations," in *2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2022, pp. 86–91.

[12] M. Rezaalipour, L. Ferretti *et al.*, "ErrorEval: An open-source worst-case-error evaluation framework for approximate computing," in *Proceedings of the 20th ACM International Conference on Computing Frontiers*, ser. CF '23. ACM, 2023, p. 393–394.

[13] J. Yang and K. S. Meel, "Rounding meets approximate model counting," in *Computer Aided Verification*, C. Enea and A. Lal, Eds. Cham: Springer Nature Switzerland, 2023, pp. 132–162.

[14] V. Mrazek, R. Hrbacek *et al.*, "EvoApprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods," in *Proc. of DATE'17*, 2017, pp. 258–261.