# Partition and Propagate: an Error Derivation Algorithm for the Design of Approximate Circuits

Ilaria Scarabottolo[1], Giovanni Ansaloni[1], George A. Constantinides[2], Laura Pozzi[1]
[1]Università della Svizzera italiana, Lugano, Switzerland
[2]Imperial College London, London, United Kingdom
ilaria.scarabottolo@usi.ch, giovanni.ansaloni@usi.ch, g.constantinides@imperial.ac.uk, laura.pozzi@usi.ch

## ABSTRACT

Inexact hardware design techniques have become popular in error-tolerant systems, where energy efficiency is a primary concern. Several techniques aim to identify circuit portions that can be discarded under an error constraint, but research on systematic methods to determine such error is still at an early stage. We herein illustrate a generic, scalable algorithm that determines the influence of each circuit gate on the final output. The algorithm first partitions the graph representing the circuit, then determines the error propagation model of the resulting subgraphs. When applied to existing approximate design frameworks, our solution improves their efficiency and result quality.

## 1 INTRODUCTION

The pervasive diffusion of resource-constrained embedded and portable systems is challenging traditional methodologies for the design of digital systems. In particular, Approximate Computing applied at circuit level is gaining increasing research interest, as it can lead to high-performance implementations coupled with reduced area and energy cost.

A promising approach to designing approximate hardware is logic simplification, which may tune the Boolean function implemented by a circuit to obtain an inexact equivalent [11], or it may focus on the pruning of gate-level netlists, identifying circuit portions that, if neglected [6, 7] or substituted [10], entail only small errors at the outputs.

For all these approximation techniques to be effective, an accurate notion of the error induced at the output by a given transformation is paramount. Our contribution falls in this field, addressing the challenge of quantifying the maximum error observed in a generic combinatorial circuit output, when some of the circuit logic gates are simplified. We propose a novel algorithm to identify a *bound* on such error, which improves on the state of the art by providing accuracy similar to that obtained through exact error computation, at a fraction of the runtime.

We achieve this goal by partitioning the original circuit into subcircuits and deriving an error-propagation model to label each gate with the maximum error induced by its removal.

As this partitioning step guarantees that we only rely on simulation for (small) subcircuits, our strategy has a remarkable positive impact on the quality of result and on the run-time required for inexact logic simplification.

Summing up, the main contributions of this work are:

- We present a circuit decomposition algorithm to quantify the error induced by inexact logic simplification, able to bypass the non-scalable complexity of exact error computation.
- We show that the obtained errors are orders-of-magnitude less conservative than those derived by maximum error estimation techniques available in the state of the art.
- We showcase the performance enhancement that our error estimation method can induce in state-of-the-art gate-level simplification frameworks. Our method enables up to 39% *further* reduction of EDAP (energy, delay and area product) for the inexact circuits derived by the GLP method [7], when guided by our error model.

## 2 RELATED WORK

Approximate Computing as a design paradigm has been attracting a large number of research efforts. While approximation can be exploited at various levels of the hardware/software stack, circuit-level AC methodologies are most related to our contribution. These techniques aim to derive inexact gate-level netlists starting from their exact counterparts; inexact circuits can then be combined to efficiently realise complex functionalities [1]. Some notable efforts in inexact circuit research focus on manually designing specific arithmetic units, such as adders [13] or multipliers [5], while others adopt a more generic approach, enabling the simplification of any combinatorial circuit [2, 4, 6, 7, 9–11]. In order for these techniques to be effective, accurate error estimation is needed to understand the effect that each proposed transformation can induce at the circuit output (for example, which node to be selected for iterative node-removal [7, 14], or which net to be substituted and simplified [10]).

Among the state of the art in error estimation methods to guide the above approximation techniques, several works [2, 7, 9, 12] present a framework where errors are derived through Monte Carlo sampling of possible inputs, and expressed as statistical measures.

The method we propose focuses instead on finding a bound for the maximum error, which is a stronger requirement that can be of primary concern in some critical applications. The state of the art in proposing a bound for maximum error — as opposed to average error estimation — consists in two categories: one of tractable complexity but producing conservative quality bounds, and one delivering exact values but exhibiting intractable complexity.

*DAC'19, June 2019, Las Vegas, Nevada USA*

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

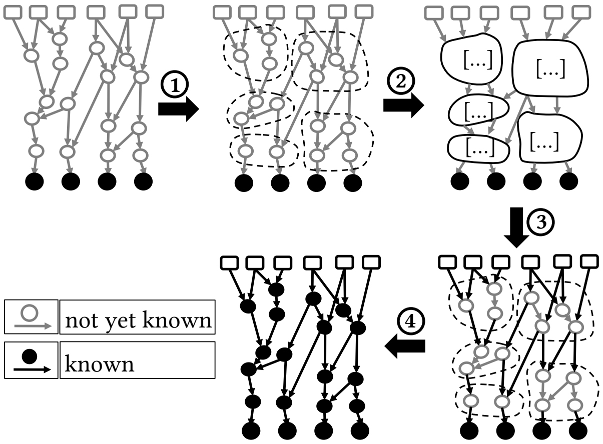

**Figure 1: The proposed P&P methodology. A graph representing a gate-level netlist is processed; error weights at this point are known only at the outputs (bit-significance). The graph is then partitioned (1), and propagation matrices are derived (2); they express the relation between output and input weights. External edges weights are then derived by propagation through subgraphs (3), and finally subgraph simulation determines weights for internal nodes (4).**

Belonging to the first category, a strategy [7] which we refer to as sum labelling models the influence of a gate as the *sum* of the influence of all its direct children gates. In practice, however, the influence of a gate removal can be considerably less than this conservative estimation, and our experimental results show that orders of magnitude can separate it from the actual error. On the other end of the spectrum, and belonging to the second category, is exact error computation. This can be performed employing whole-circuit simulation, as in [6], or using SAT solvers, as in [12]; however, the complexity of these methods is exponential in the number of circuit inputs, and therefore necessarily becomes intractable at some point.

The present work, called Partition and Propagate (P&P), overcomes the intractability of exact error computation, by partitioning the circuit and employing simulation only on small *subcircuits*, and at the same time strongly improves on the accuracy obtained by tractable strategies such as sum labelling [7], as demonstrated by experimental results.

## 3 METHODOLOGY

In the following sections, we formally state the problem we address; then, we describe the stages of our workflow (also depicted in Figure 1), namely *graph partitioning*, *derivation of the propagation matrices*, *propagation* and *subgraph simulation for internal nodes.*

### 3.1 Problem Formulation

A combinatorial circuit can be represented as a Direct Acyclic Graph $G(N, E)$, where each node $n_i \in N$ represents a single output Boolean gate and each edge $e(n_i, n_j) \in E$ represents a connection between nodes such that the output of $n_i$ is used by $n_j$.

The purpose of this work is to label the graph with weights $w(n_i)$ for each gate $n_i \in N$. The weight of an edge is equal to that of its destination node: $w(e(n_i, n_j)) = w(n_j)$ for all $e \in E$.

Weights represent the influence of a node on the circuit output, in terms of a bound on the maximum difference from the exact result that can be observed if $n_i$ is removed from the circuit and its output is set to a constant value.

The result of a circuit computation is captured by function

$$f : \mathbb{B}^n \rightarrow \mathbb{Z} \tag{1}$$

mapping the Boolean vector of the circuit primary outputs into an integer, corresponding to bit-significance weighting for arithmetic circuits.

Given a graph $G(N, E)$ representing the exact circuit, and given graph $G_i(N \backslash \{n_i\},\ E \backslash \{e(u, v) \mid u = n_i \vee v = n_i\})$ representing the approximate one, obtained by removing node $n_i$, for all $k = 1..2^{|I|}$ possible input combinations the two circuits will generate the Boolean vectors $\boldsymbol{o}^k$ and $\boldsymbol{o}_i^k$ of their primary outputs. The exact weight of $n_i$ is, then, their maximum difference:

$$w(n_i) = \max_k |f(\boldsymbol{o}^k) - f(\boldsymbol{o}_i^k)| \tag{2}$$

### 3.2 Graph partitioning

The strategy we adopt to obtain a scalable framework to derive such weights is to partition the graph into smaller subgraphs that can be separately simulated. We define a *partition* function

$$p : G(N, E) \mapsto S \tag{3}$$

where $S$ is a set of subgraphs $s(N_s, E_s)$, $N_s \subseteq N$, $E_s \subseteq E$. Each $n_i \in N$ is assigned to exactly one subgraph $s(N_s, E_s) \in S$, and $\bigcup_s N_s = N$. All edges $\{e(n_i, n_j) \in E \mid n_i \in N_s \wedge n_j \in N_s\}$ belong to $E_s$. Figure 1 (step 1) shows an example of a generic graph $G$ partitioned into five subgraphs. We define *external edges* $\{e(u, v) \mid u \in N_i \wedge v \in N_j, i \neq j\}$ those linking two different subgraphs. A subgraph $p(N_p, E_p)$ is *parent* of subgraph $q(N_q, E_q)$ if there exists at least one external edge $\{e(u, v) \mid u \in N_p \wedge v \in N_q\}$.

We explain in Section 3.6 the algorithm we propose for partitioning. For now, we assume a partition has already been performed.

### 3.3 Derivation of the propagation matrix

After graph $G(N, E)$ has been partitioned, each subgraph is analysed to determine how the weights of its inputs can be derived as a function of those of its outputs.

Each subgraph has a set of inputs $I_s = \{e(u, v) \mid u \notin N_s, v \in N_s\}$ and a set of outputs $O_s = \{e(u, v) \mid u \in N_s, v \notin N_s\}$. Weights of the inputs are obtained as functions of output weights by observing the subgraph truth table. An example is provided in Figure 2, where a subgraph of two inputs, edges $a_1$ and b (the two edges coming from node B are actually indistinguishable, as they correspond to the same bit value, and hence are referred to as a single one), and two outputs, C, D is depicted; weights of the two outputs are labelled $w_C$ and $w_D$.

In particular, Figure 2b illustrates how $w_b$ is obtained: a pairwise comparison is performed between input tuples that differ only for the value of bit b. If the integer value of the output vector

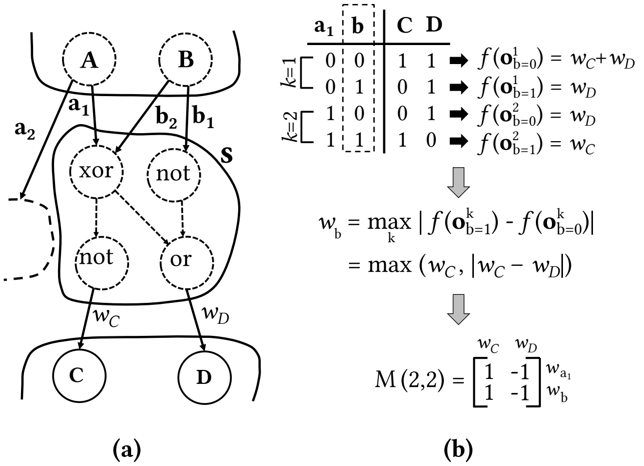

**Figure 2: Propagation matrix derivation for an example subgraph. The figure reports its truth table, and differences are computed for $w_b$. The process is repeated twice (once for each distinguishable input) to obtain the complete matrix.**

$f(\boldsymbol{o})$ is strictly monotonically increasing with subgraph output bits C and D, then $w_b$ is set to the maximum recorded difference $\max_k |f(\boldsymbol{o}^k_{b=1}) - f(\boldsymbol{o}^k_{b=0})|$ over the $k = 1..2^{I_s-1}$ comparisons (2, in this basic example), and it is expressed as a linear combination of the output weights (in the example, we assume $|w_C - w_D|$ is greater than $w_D$).

We can safely set $w_b$ to the *difference* of output bits weights because the strict monotonicity of $f(\boldsymbol{o})$ guarantees that errors at the subgraph output will propagate in the same way (*i.e.* with the same polarity) to the graph primary outputs. If, on the contrary, $f(\boldsymbol{o})$ is not monotonically increasing with bits C or D, the direction of each of these bit variations could be reverted in lower computations and, hence, we are forced to set the input weight to the sum of all flipped output bit weights. In the example, this would result in weight $w_b$ being set to $w_C + w_D$. Note that a single non-monotonic subgraph in the path to the primary outputs is sufficient for potential error underestimation; therefore, information on non-monotonicity must be retained for upper subgraphs.

The same process is repeated for input $a_1$, and the results are stored in a *propagation matrix* $M_s(|I_s|,\ |O_s|)$, where the $i$-th row contains the coefficients (0, 1 or -1) of the linear combination of the output weights for input $i$. Weights of the subgraph inputs can then be calculated as:

$$\begin{bmatrix} w_{a_1} \\ w_b \end{bmatrix} = M \begin{bmatrix} w_C \\ w_D \end{bmatrix} \tag{4}$$

Propagation matrices are derived for each subgraph, then used to propagate subgraph output weights to their inputs.

## 3.4 Propagation

The missing step at this point is to propagate weights from the *inputs* of generic subgraphs to the *outputs* of their *parent* subgraph(s). For example, looking again at Figure 2, weights of nodes A and B must now be derived from weights of edges $a_1$, $a_2$ and b.

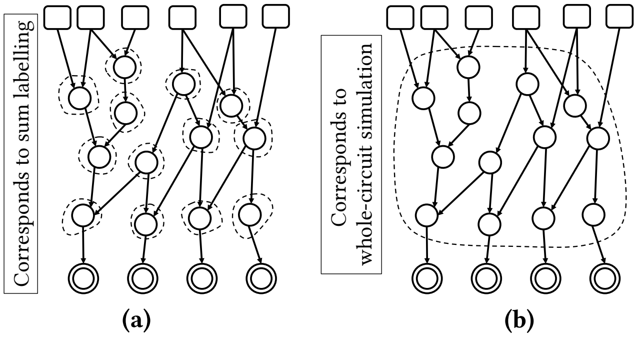
Corresponds to sum labelling
(a)
Corresponds to whole-circuit simulation
(b)

**Figure 3: Two end-of-the-spectrum partitions: a) each gate corresponds to one subgraph, b) the whole circuit corresponds to a single subgraph.**

For a generic subgraph output, either all its children belong to the same subgraph (as for node B of Figure 2), or children nodes are distributed in different subgraphs (as for node A). Derivation for the first case is trivial: $w_B$ is equal to $w_b$. However, if a node has children belonging to different subgraphs, its weight must conservatively be computed as the *sum* of the corresponding external edges weights ($w_{a_1} + w_{a_2}$ for $w_A$ in the example), since we cannot resort to a *single* truth table to compute a less conservative weight.

## 3.5 Subgraph simulation for internal nodes

Once external edges and subgraph outputs are labelled, we populate each subgraph with internal node weights. In this phase, we do employ exhaustive simulation, as in [6], but applied to *each subgraph separately*, where the crucial difference is that their number of inputs $|I_s|$ is much smaller than that of the whole circuit $|I|$. A SAT-solver based exact weight derivation could also be used in this step, leading to the same conclusion: the number of inputs of subgraphs is now limited, and hence the problem becomes tractable.

## 3.6 Partitioning criteria

Different partition choices for a given graph $G$ will impact two aspects: the accuracy of the obtained weights on one side, and the feasibility of subgraph simulation on the other.

Figure 3 depicts two extreme partitions: at one end, each gate corresponds to one subgraph. This leads to a *feasible* partition (each subgraph has a limited number of inputs – namely two, at most, for up to two-inputs gates – and hence can be simulated) but will be the worst in terms of resulting weight accuracy, because it corresponds to resorting to the sum labelling algorithm. In fact, with this partition, all gate fanouts are external edges and the propagation step (Section 3.4) is conservatively forced to adopt sum labelling for all of them, when deriving weights of subgraph outputs.

At the other extreme, the whole circuit corresponds to a single subgraph. This will lead to the best partition in terms of resulting weight accuracy (all children of the same gate belong to the same sugraph, and hence get subsumed in a single propagation matrix), but will be generally infeasible, because it corresponds to whole-circuit simulation.

To find a suitable partition trading off these concerns, we consider two main aspects. First, *feasibility*: the number of inputs $|I_s|$ for each subgraph $s$ has to be small, in order to allow simulation of all the $2^{|I_s|}$ possible input combinations.

Secondly, as illustrated in Section 3.4, it is advantageous that *all children of any given node belong to the same subgraph*, so that gate fanouts are included into the same subcircuit simulation and sum labelling employment is unnecessary. Hence, for each $n', n''$ children of the same node:

$$\forall N', N'' \in N.\ n' \in N' \wedge n'' \in N'' \ \Rightarrow N' = N''. \qquad (5)$$

Note that state-of-the-art graph partitioning approaches for logic synthesis (such as, for example, KL-cuts [3]), do not consider the condition expressed in Equation 5 above, and hence are not suitable for this problem; therefore, we propose and implement a different algorithm. The partitioning algorithm we propose labels graph nodes with subgraph IDs, by first assigning the same subgraph ID to all children of the same node, iteratively merging subsets with same ID to 1) honour the condition for all nodes, and 2) guarantee an acyclic partition. In a second traverse, we merge together subgraphs for which $|I_{s+t}| \leq \max\{|I_s|, |I_t|\}$, *i.e.*, the number of inputs of the resulting subgraph does not increase with respect to the largest one among its components. This is done to reduce the number of propagation matrices to be derived, without increasing the computational cost for any of them.

This algorithm creates a first phase partition, which is exemplified in Figure 4a. However, a partition thus created does not guarantee feasibility, because not all subgraphs necessarily will have a limited number of inputs (the feasible number of inputs is set to 3 for the simple example in the figure, and an infeasible subgraph exists at the top-left). Hence, to recover the feasibility property, we adopt the simple strategy of further partitioning exactly those subgraphs that are infeasible using the one-node-one-subgraph extreme partition scheme, as exemplified in Figure 4b (the percentage of nodes ending up in infeasible subgraphs after first phase is presented in our experimental section).

Thus, this two-stage partitioning strategy generates 1) a *feasible* partition, where 2) a high number of node fanouts are included in the same subgraph, and hence are subsumed in a single propagation matrix.

### 3.7 Time complexity analysis

For the sake of readability, in this section we will refer to the cardinality of a set with the name of the set itself.

For a graph with $N$ nodes, partition $p$ is obtained through a first graph traverse, of cost $N$, for children subgraph ID assignment. The resulting partition is then explored to guarantee that, if there exist subgraphs leading to cycles, these are merged in a single one. This step has worst case complexity $O(S + n_{EE})$, where $S$ is the number of resulting subgraphs *before* further partitioning due to infeasibility, and $n_{EE}$ the number of external edges. After the second subgraph-merging traverse of cost $S$, infeasible subgraphs are split into single-gate subgraphs in another linear step. Therefore, partitioning has complexity $O(N + S + n_{EE})$.

Propagation, then, implies subgraph simulation for all subgraph inputs, and for all internal gates. Time complexity of this phase strongly depends on the partition: for the extreme partition of Figure 3b, it would be $O(N2^I)$, while for that of Figure 3a only $O(4N) = O(N)$, since all gates have 2 inputs and, hence, 4 possible input patterns. For generic partitions, the time complexity is $O(\sum_{s=1}^{S} N_s 2^{I_s})$. Since we bound $I_s$ to a threshold $T_I$, the exponential term can be expressed as a constant $C = 2^{T_I}$, leading to $O(C\sum_{s=1}^{S} N_s) = O(CN) = O(N)$.

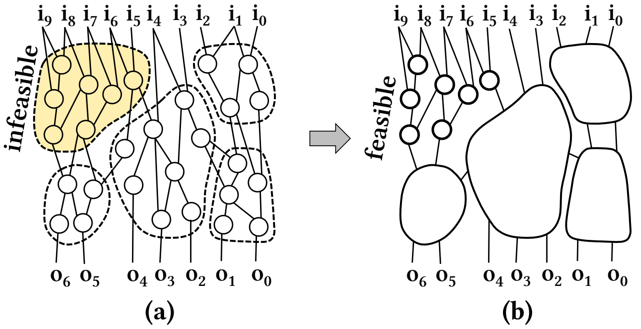

**Figure 4: a) A first phase partition with one infeasible subgraph (top-left), b) A second phase feasible partition, with the infeasible subgraph further partitioned into single-gate subgraphs.**

In conclusion, the overall time complexity of the proposed algorithm is $O(N + S + n_{EE})$, which depends on the chosen partition parameters but is still remarkably lower than approaches that resort to simulation of all inputs or SAT-solver based exact weight derivation. In our experiments, $C$ is set to $2^{10}$, while the number of input combination for processed circuits ranges between $2^{16}$ and $2^{64}$. A widespread alternative to exhaustive simulation is Monte Carlo selection of a subset of possible input patterns; the resulting simulation has then a complexity of $O(MN)$, given $M$ the cardinality of such subset. However, to obtain accurate estimations, $M$ must be considerably larger than $C$ (for example, $2^{17}$ in [9], $2^{20}$ in [2] and more than $2^{22}$ in [7]). Moreover, our method provides a *bound* for maximum error, while MC only provides an approximation of maximum error.

## 4 EXPERIMENTAL EVALUATION

### 4.1 Partition and Propagate performance

We assessed the performance of our methodology on a diverse set of benchmark circuits, specified in the VHDL language. Their characteristics are described in Table 1, along with the critical path constraints employed in their synthesis. We used Synopsys Design

**Table 1: Benchmark netlists.**

| Benchmark | # of gates | I/O bitwidth | critical path (ns) |
|---|---|---|---|
| 8-bit adder | 115 | 16/9 | 0.5 |
| 8-bit absolute difference | 245 | 16/9 | 0.1 |
| 32-bit adder | 475 | 64/33 | 2.0 |
| 16-bit butterfly | 485 | 32/33 | 2.0 |
| 8-bit multiplier | 685 | 16/16 | 2.0 |
| 8-bit binomial squared | 946 | 16/18 | 5.0 |

**Table 2: Influence of threshold $T_I$ on resulting partitions, in terms of number of resulting subgraphs, infeasible subgraph ratio, and average distance from exact weights (when available).**

| $T_I$ | 8-bit adder | | | 8-bit abs. diff. | | | 32-bit adder | | | 16-bit butterfly | | | 8-bit multiplier | | | 8-bit bin. sq. | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | n. sg. | inf. % | avg dist. | n. sg. | inf. % | avg dist. | n. sg. | inf. % | avg dist. | n. sg. | inf. % | avg dist. | n. sg. | inf. % | avg dist. | n. sg. | inf. % | avg dist. |
| 10 | 7 | 0.0 | 0.00e0 | 17 | 0.0 | 1.43e2 | 31 | 0.0 | - | 15 | 0.0 | - | 190 | 9.6 | 2.34e5 | 8 | 0.0 | 4.36e3 |
| 5 | 7 | 0.0 | 0.00e0 | 135 | 44.1 | 2.26e3 | 31 | 0.0 | - | 15 | 0.0 | - | 190 | 9.6 | 2.34e5 | 272 | 16.1 | 1.27e7 |
| 2 | 63 | 44.3 | 3.47e2 | 163 | 55.6 | 2.69e3 | 231 | 36.3 | - | 430 | 78.9 | - | 340 | 30.4 | 6.58e5 | 648 | 57.0 | 3.79e9 |

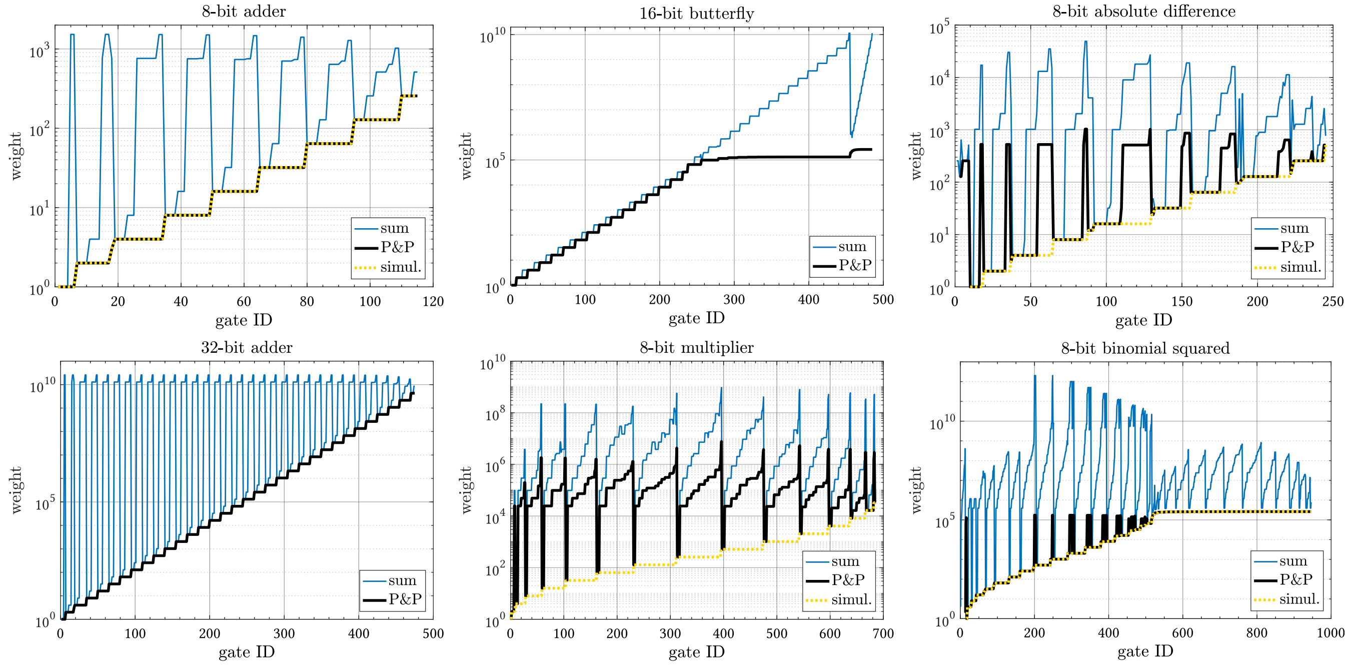

**Figure 5: Comparison of the gate weights computed using sum labelling [7], Partition and Propagate and, for circuits with lower input number, whole-circuit simulation [6].**

Compiler as a synthesis tool, targeting a 40nm technology library, while SIS [8] was used to carry out the simulation of subcircuits, as well as exact error computation for circuits with up to 16 inputs. Benchmarks were first synthesised to netlists only comprising inverters and 2-inputs NAND and NOR gates; then, these netlists were fed to our tool P&P to compute gate weights. Our framework can process netlists composed by any set of logic gates; nonetheless, the use of larger, compound, gates may adversely impact flexibility, because, as weights are assigned on a per-gate bases, it would force simplification frameworks to monolithically discard or retain large circuit portions.

Our strategy allows one to trade-off the accuracy of weights and computational effort by tuning, during the partitioning phase, threshold $T_I$, the maximum number of inputs allowed in a subgraph. Showcasing the impact of this parameter, Table 2 reports the number of obtained subgraphs, the infeasible subgraph ratio (*i.e.*, the number of gates assigned to infeasible subgraphs over the total number of gates), and the average distance from exact weight, for $T_I = 10$, $T_I = 5$ and $T_I = 2$. As can be expected, lower thresholds result in more and smaller subgraphs, and consequently in more conservative weights, as more subgraphs are split into one-node-one-subgraph partitions. A value of $T_I = 10$ obtains high quality weights, while the resulting computation time remains within seconds to minutes, and it is the value that we choose for all further experiments reported in this section.

In Figure 5 we compare the performance of P&P, in terms of maximum error estimation accuracy, with state of the art methods: sum labelling [7], a conservative and low-complexity strategy, and exact error computation, which has exponential complexity and can be obtained by whole-circuit simulation [6] or by a SAT formulation [12]. It can be seen that P&P retrieves much less conservative gate weights with respect to sum labelling (for example, seven orders of magnitude less for gate with ID 200 in the 8-bit binomial squared).

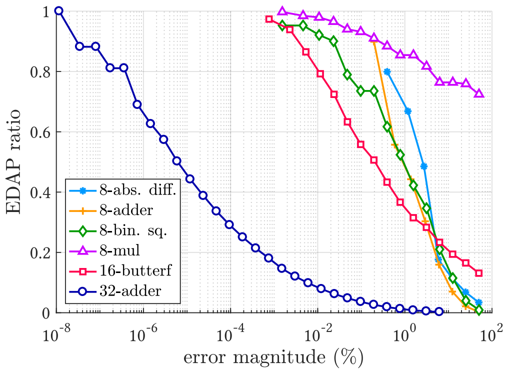

**Figure 6: EDAP (Energy-Delay-Area Product) gains for all benchmarks.**

**Table 3: Average EDAP ratio of GLP-pruned circuits guided by either sum [7] or P&P labelling.**

| Benchmark | average EDAP ratio | |
|---|---|---|
| | GLP + sum [7] | GLP + P&P |
| 8-bit adder | 0.503 | 0.394 |
| 8-bit absolute difference | 0.574 | 0.443 |
| 32-bit adder | 0.510 | 0.333 |
| 16-bit butterfly | 0.507 | 0.507 |
| 8-bit multiplier | 0.885 | 0.877 |
| 8-bit binomial squared | 0.939 | 0.553 |

P&P weights in fact approach – or even coincide with – those obtained with exact error computation.

Because of its low computational complexity, P&P runtime is very short. For example, to label all gates of the 32-bit adder it took only 18 seconds, while SAT-solver exact error computation of [12] takes 8 minutes for a circuit with similar gate count (≈ 400), and a Monte Carlo simulation with 1 million inputs as in [2] takes 11 seconds for each design point, which would result in more than an hour to label all circuit gates.

In conclusion, P&P combines graceful scaling with accurate error estimation.

## 4.2 Performance of approximate circuits

The improved accuracy in gate weights achieved by P&P has a significantly positive impact on the subsequent simplification of inexact circuits. To explore this effect, we reimplemented the approximation strategy proposed in [7], called Gate Level Pruning (GLP), which iteratively selects gates to be removed until the error constraint is violated, starting from gates of lower weight. These pruned netlists were then re-synthesised, using the entire set of library gates, to retrieve their Energy Delay Area Product (EDAP) that we use in Figure 6 as a figure of merit.

Figure 6 illustrates that P&P algorithm leads to high-performance approximate circuits, resulting in large EDAP gains for small error thresholds. As an example, for the case of the 8-bit binomial squared benchmark, an EDAP improvement of 50% was achieved for an error magnitude of slightly less than 1%, where error magnitude is a measure of absolute error, expressed as a percentage of the maximum circuit output.

Further showing the benefit of our error derivation methodology, Table 3 reports the average EDAP ratio obtained for each benchmark when GLP is guided by sum labelling, as actually done in [7], and when it is guided by P&P labelling instead. P&P generally induces large further reductions in obtained EDAP values, of up to 39% for the 8-bit binomial squared case, and of 14% on average.

# 5 CONCLUSIONS

This work presents a novel algorithm for gate-level error determination, which identifies the maximum error observed on the output when a gate is removed from a combinatorial circuit. Our approach overcomes the main limitations of previous methods for quantifying such errors, by adopting a divide-and-conquer strategy that results in a generic and scalable algorithm. When applied to AC design methods, the proposed algorithm upgrades their awareness of error-propagation, resulting in higher result quality.

Future work will focus on investigating a larger spectrum of partitioning criteria, as well as on the adaptation of our method to probabilistic error measures.

# REFERENCES

[1] J. Castro-Godínez, S. Esser, M. Shafique, S. Pagani, and J. Henkel. 2018. Compiler-Driven Error Analysis for Designing Approximate Accelerators. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition.* 1–6.

[2] S. Hashemi, H. Tann, and S. Reda. 2018. BLASYS: Approximate Logic Synthesis Using Boolean Matrix Factorization. In *Proceedings of the 55th Design Automation Conference.* 55:1–55:6.

[3] O. Martinello, F. S. Marques, R. P. Ribas, and A. I. Reis. 2010. KL-Cuts: A new approach for logic synthesis targeting multiple output blocks. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition.* 777–782.

[4] J. Miao, A. Gerstlauer, and M. Orshansky. 2013. Approximate logic synthesis under general error magnitude and frequency constraints. In *Proceedings of the International Conference on Computer Aided Design.* 779–786.

[5] S. Rehman, W. El-Harouni, M. Shafique, A. Kumar, and J. Henkel. 2016. Architectural-space exploration of approximate multipliers. In *Proceedings of the International Conference on Computer Aided Design.* 1–8.

[6] I. Scarabottolo, G. Ansaloni, and L. Pozzi. 2018. Circuit Carving: A Methodology for the Design of Approximate Hardware. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition.* 545–550.

[7] J. Schlachter, V. Camus, K. V. Palem, and C. Enz. 2017. Design and Applications of Approximate Circuits by Gate-Level Pruning. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 25, 5 (Feb. 2017), 1694–1702.

[8] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli. 1992. SIS: A system for sequential circuit synthesis. (1992).

[9] S. Su, Y. Wu, and W. Qian. 2018. Efficient Batch Statistical Error Estimation for Iterative Multi-level Approximate Logic Synthesis. In *Proceedings of the 55th Design Automation Conference.* 54:1–54:6.

[10] S. Venkataramani, K. Roy, and A. Raghunathan. 2013. Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition.* 1367–1372.

[11] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan. 2012. SALSA: systematic logic synthesis of approximate circuits. In *Proceedings of the 49th Design Automation Conference.* 796–801.

[12] R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan. 2011. MACACO: Modeling and analysis of circuits for approximate computing. In *Proceedings of the International Conference on Computer Aided Design.* 667–673.

[13] R. Ye, T. Wang, F. Yuan, R. Kumar, and Q. Xu. 2013. On reconfiguration-oriented approximate adder design and its application. In *Proceedings of the International Conference on Computer Aided Design.* 48–54.

[14] Z. Zhang, Y. He, J. He, X. Yi, Q. Li, and B. Zhang. 2018. Optimal Slope Ranking: An Approximate Computing Approach for Circuit Pruning. In *Proceedings of the 2018 IEEE International Symposium on Circuits and Systems.* 1–4.