This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2019.2946254, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

Parallel Combinational Equivalence Checking

Vinicius N. Possani, *Member, IEEE,* Alan Mishchenko, *Senior Member, IEEE*
Renato P. Ribas, *Member, IEEE,* Andre I. Reis, *Senior Member, IEEE*

***Abstract*—Combinational equivalence checking (CEC) has been widely applied to ensure design correctness after logic synthesis and technology-dependent optimization in digital IC design. CEC runtime is often critical for large designs, even when advanced techniques are employed. Three complementary ways for enabling parallelism in CEC are proposed, addressing different design and verification scenarios. Experimental results have demonstrated speedups up to 63x when comparing the proposed approach to a single-threaded implementation of similar CEC engine. A practical impact of such a speedup, for instance, is the runtime reduction from 19 hours to only 18 minutes when checking equivalence of AIGs comprising more than twenty million nodes. Therefore, the proposed solution presents great potential for improving current EDA environments.**

***Index Terms*—Digital IC Design, Verification, Combinational Equivalence Checking, Graph Partitioning, Parallel Computing.**

I. Introduction

FAST and scalable techniques for combinational equivalence checking (CEC) are essential in modern electronic design automation (EDA) environments. In a typical scenario, the logic function implemented by an optimized digital integrated circuit (IC) is checked for equivalence to the original specification after multi-level logic synthesis [1]. Moreover, scalable CEC techniques are quite useful in several key logic synthesis processes which depend on efficient computation of equivalence classes of internal circuit nodes. A non-exhaustive list of those tasks is the following:

- Removal of functionally equivalent logic in the design during logic synthesis and optimization;
- Computation of structural choices, enabling circuit area and signal delay improvement after technology mapping step [2];
- Sequential equivalence checking based on register and signal correspondence [3], [4];
- Bridging circuits for the implementation and the specification in engineering change orders (ECOs) [5];
- A number of utility packages, *e.g.* node name transfer across netlists before and after synthesis.

In recent years, equivalence checking has become more critical due to the increasing in complexity of current and upcoming system-on-chip (SoC) and VLSI designs. To demonstrate the complexity of CEC when dealing with large designs, consider one instance of the problem for checking equivalence between the original circuit description comprising 14 million nodes and a synthesized version comprising 9 million nodes being both represented through AND-inverter graphs (AIGs). These circuits were checked using the $\&cec$ command in ABC, which is an industrial-strength academic tool for logic synthesis and formal verification [6]. ABC took more than 24 hours to prove equivalence, making it clear that such a verification becomes harder as the design size increases, so reinforcing the necessity for improved CEC to make computer-aided design (CAD) tools more scalable.

In order to enable the next rounds of technology innovation, there is a demand for massively parallel EDA tools running on the cloud [7], [8]. Considering this scenario, traditional EDA algorithms and data structures, in particular, those dealing with CEC, need to be rethought to benefit from parallel computing platforms. In typical EDA environment, algorithms work on a sparse graph representing the circuit. It is often challenging to exploit parallelism of graph-based algorithms due to the irregular nature of graphs implemented using pointer-based data structures. Besides, several CEC approaches are based on binary decision diagram (BDD) [9]–[11] and hybrid BDD/SAT-based engines [12], which are less scalable when running into single thread and so harder to parallelize than those based on simulation and Boolean satisfiability (SAT) [13], [14].

In this paper, we are unlocking massive parallelism for CEC by applying graph (miter) partitioning in order to balance data sharing and data independence during SAT solving. The paper proposes two models of parallelism that can be applied separately or combined, leading to a third hybrid model that allows to fully exploit the power of parallel environments. Our parallel CEC is based on the state-of-the-art CEC engine available in ABC, which exploits the synergy between logic simulation and SAT [14].

The main contributions of this work are the following:

- Three novel models are introduced to enable massive parallelism for speeding up two crucial time-consuming CEC tasks, mitering and SAT sweeping.
- The proposed parallel CEC engine handles large designs comprising millions of AIG nodes, and scales to many threads unlocking the potential of parallel environments available through cloud computing.
- Experimental results have shown significant runtime improvement when comparing to both single-threaded ABC and parallel commercial CEC engines. In some cases, the proposed solution reduces the CEC runtime from more than one day to only a few minutes/hours.
- Proposed models are based on simple principles making them easy to reproduce and deploy in a standard EDA flow. Therefore, several other important tasks that depend

V. N. Possani, R. P. Ribas and A. I. Reis are with the Institute of Informatics, Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil. The authors can be contacted at: vnpossani@inf.ufrgs.br; andreis@inf.ufrgs.br; rpribas@inf.ufrgs.br.

A. Mishchenko is with the Department of EECS, University of California, Berkeley, USA. The author can be contacted at: alanmi@berkeley.edu.

Manuscript received March 25, 2019.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2019.2946254, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

on CEC can benefit from the speedup enabled by the proposed scalable solution.

The rest of this paper is organized as follows. Section II presents a brief review of techniques used in modern CEC engines. Related work is discussed in Section III. Section IV describes the proposed approach for accelerating CEC. Experimental results are discussed in Section V. The conclusions are outlined in Section VI.

II. Preliminaries

This section presents a set of definitions related to the main concepts and techniques involved in the proposed approach for parallel CEC.

A. AND-Inverter Graph

AND-inverter graph is a directed acyclic graph used as data structure in logic synthesis. AIG is a homogeneous circuit representation comprising four types of nodes: constants, primary inputs (PI), primary outputs (PO) and two-input AND (AND2) operators. Sequential elements such as latches and flip-flops can be viewed as special nodes or pseudo-PI/PO. A graph edge can present an optional attribute depending on whether the corresponding signal is complemented.

The set of nodes connected to the inputs of a given AIG node n is called the *fanin* of n. Analogously, the set of nodes connected to the outputs of n is called the *fanout* of n. If there is a path from node n to n', then n is in the *transitive fanin (TFI)* of n' and n' is in the *transitive fanout (TFO)* of n. The *TFI cone* of a given node n includes n and all those nodes in the transitive fanin of n towards the primary inputs of the AIG. The *TFO cone* of n is analogous, including n and all those nodes in the transitive fanout of n towards the primary outputs of the AIG. *Structural hashing* is a technique used to ensure that a given AIG does not have duplicated AND2 nodes with exactly the same pair of fanin, even under permutation of the AND2 inputs [15].

B. Boolean Satisfiability

Boolean satisfiability is the decision problem to determine whether there exist an assignment to the input variables that makes the output of a given Boolean formula F evaluates to $true$. If such an assignment exists, then F is $satisfiable$ (sat). Otherwise, F is $unsatisfiable$ ($unsat$). Conventionally, SAT problem instances are represented by a formula F in the conjunctive normal form (CNF). A $literal$ is an instance of a given Boolean variable x, e.g. x (positive) or $\neg x$ (negative). A $clause$ is a disjunction of literals and a $cube$ is a conjunction of literals. SAT solvers are tools implementing advanced techniques for automatically deciding whether a given formula F is sat or $unsat$. Common SAT techniques applied nowadays are *lookaheads* [16], *conflict-driven clause learning* (CDCL) [17] and *incremental SAT solving* [18], [19].

C. Mitering

A *miter* M composes two circuits under verification C_1 and C_2 into a single circuit by pair-wise connecting the inputs with the same name and by pair-wise comparing the outputs with the same name using exclusive-OR (EXOR) operator [20]. At the top of the miter, an OR operator connects all the outputs of EXORs, representing the output of the comparator for the primary outputs, as shown in Fig. 1.

In practice, M can be represented by a multi-output AIG without the top OR operator so that each output is a separate decision problem. This way, the AIG is viewed as an instance of a multi-output SAT problem that can be easily converted to CNF form by using the Tseitin transformation [21], and then given to the SAT solver. If the SAT solver returns $unsat$, all pairs of outputs under comparison are equivalent. Otherwise, the solver returns sat, *i.e.*, at least one pair of outputs is different. This process for proving equivalence using miter and SAT solving is also referred as *mitering*.

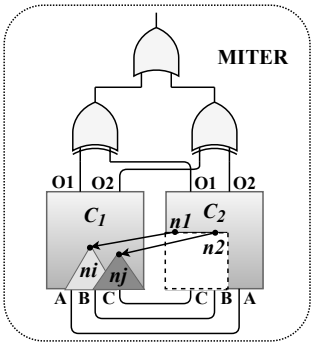


Fig. 1. Miter structure with reduced logic by merging equivalent internal nodes.

D. BDD and SAT Sweeping

The techniques known as *BDD sweeping* [12] and *SAT sweeping* [22] are used to detect functionally equivalent nodes internally in AIG or miter. In this approach, pairs of internal nodes are checked for equivalence in a topological order. If the equivalence is proved, the corresponding nodes can be merged to simplify the miter, as shown int Fig. 1. BDD and SAT sweeping are useful since to prove equivalence by directly constructing a BDD of a miter for the entire circuits under verification is often not practical [23]. In many cases, directly applying BDD construction and SAT solving for checking equivalences require a lot of memory and computation resources.

E. Logic Simulation

Logic simulation is the process of applying random values to the circuit PIs and propagate these values towards the circuits POs in order to asses the logic behavior of internal nodes and/or POs. This is a common technique used to quickly detect non-equivalent nodes, helping to reduce the number of SAT calls during SAT sweeping. Typically, random vectors and counter-examples returned by SAT solvers are used as input

patterns for simulation [13], [22]. A *counter-example* C_{ex} is an assignment of input variables of the Boolean formula F representing the SAT problem instance, in this context, proving that a pair of nodes is not equivalent. This way, simulation is used to group potential-equivalent nodes into classes whereas SAT solving is used for checking equivalences among nodes belonging to the same class.

III. Related Work

In formal verification, model checking aims to exhaustively verify a software/hardware system against to the target specification in order to ensure the correct system behavior [24], [25]. During the verification process, several system properties must to be checked such as *liveness* and *safety* [26], where CEC is a particular instance of a safety property. Early symbolic model checking methods were mainly based on BDDs [23] whereas modern approaches are based on SAT solvers, becoming more scalable for current industrial-size instances [27]–[29]. Currently, SAT solvers play an important role in formal verification as a whole so that advances in parallel SAT solving can improve the performance of several verification tools including the CEC ones which are of particular interest in this work. In this sense, we present a brief review on portfolio and divide-and-conquer techniques employed for speeding up SAT solving in parallel environments [30].

Portfolio-based SAT solvers aim to increase the solution-space exploration by executing multiple solvers in parallel in order to solve the same CNF formula F with different configurations or even with different SAT solvers [31]. Since SAT solvers work heuristically and tend to present significant runtime variation for different problem instances, the portfolio technique tries to take advantage of the best characteristics of several solvers according to the given instance. Clause exchange can be considered among the SAT solvers for avoiding re-computation and improving runtime. Some examples of parallel portfolio-based solvers are ManySat [32], Plingeling [33] and HordeSat [34]. In [35], dual SAT can be considered as a kind of portfolio-based method by running multiple SAT solvers on the original CNF and on its dual version.

Divide-and-conquer parallel SAT solvers decompose a given CNF formula F into disjoint sub-problems trying to find out workload balance among several workers. The dived-and-conquer approach is intuitive and it has been addressed by several previous works, some of them are addresses in [36]–[38]. However, it is challenging to decompose the given problem into pieces with similar size and computational complexity. Recent advancing in this direction is the cube-and-conquer paradigm [39] that uses a lookahead solver to partition a SAT instance in many subproblems by creating cubes and then uses a CDCL solver to solve those subproblems. The partitions are solved independently and can be naturally processed in parallel leading to linear speedups in several real-application instances.

When it comes to dedicated CEC engines, in [40], a different method called EQUIPE was proposed for parallelizing CEC based on a hybrid solution that combines CPU and GPU. The authors exploit parallelism in GPU to perform signature-based analysis and structural matching in order to minimize the number of SAT calls. The method evaluates the circuits under verification trying to prove speculatively the equivalence between internal nodes. When the equivalence cannot be proved by the GPU-based solution, a SAT solver is then executed in the host processor (CPU) in order to check equivalence of individual nodes. However, even when using 14 GPU-cores and 4 CPU-cores, the speedup provided by EQUIPE is limited to a factor of three compared to the non-parallel CEC engine available in ABC tool.

There is still a set of recent methods, not necessarily parallel, dedicated to the verification of arithmetic circuits by applying computer algebra techniques [41]–[45]. In general, these algebraic methods have presented significant advantages when compared to SAT-based solution for checking Galois Field arithmetic circuits and integer arithmetic circuits, so defining a different class of verification tools.

IV. Proposed Parallel CEC

In this work, we propose three different strategies to speedup CEC by enabling massive parallelism in the ABC $\&cec$ engine [6]. As discussed before, the conventional CEC solves many SAT problems represented as a miter. The proposed approach relies on graph partitioning aiming to exploit data independence during miter simplification and solving.

First of all, we dissect the CEC engine available in ABC tool, presenting its main components and investigating the most promising points to exploit parallelism. In the sequence, we define the proposed strategy for graph partitioning followed by two models for enabling parallel CEC by partitioning the main miter and the temporary miters constructed during SAT sweeping. Finally, we present the combination of both strategies can be combined in a hybrid model that allows to fully exploit the power of parallel computing.

A. CEC Engine in ABC Tool

Typically, modern CEC engines alternate between miter solving to prove equivalences of outputs and miter simplification to prove equivalence of internal nodes. Simulation and SAT sweeping are used for gradually simplifying the miter complexity. Therefore, for the sake of simplicity, we have adopted the terms *main miter* and *internal miter* when referring to the data being processed in different steps of the CEC engine core.

- Let M_m be the *main miter*, referring to the miter created once at the beginning of the verification process. M_m comprises all the logic of the two circuits under comparison. During the CEC, M_m is gradually simplified by merging internal equivalent nodes from both circuits.
- Let M_i be the *internal miter*, referring to the miter created temporarily at each integration of SAT sweeping in the CEC core. M_i comprises subgraphs from M_m, *i.e.,* $M_i \subset M_m$, representing a set of CEC subproblems for determining those internal equivalent nodes.

Fig. 2 shows a scheme of the CEC core implemented in the ABC command $\&cec$, which is an improved version of the method presented in [14]. This engine is based on the SAT solver *MiniSat* [18] and employs several techniques

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2019.2946254, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

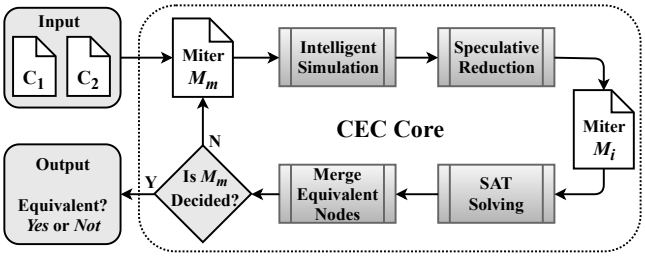


Fig. 2. CEC engine of the ABC command &*cec*.

introduced in previous work such as *intelligent simulation* and *functionally reduced AIGs (FRAIGs)* [46].

Initially, intelligent simulation is applied to group nodes that appear to be equivalent into classes. The input patterns used for simulation are computed based on SAT counter-examples C_{ex} that contribute to distinguish non-equivalent nodes belonging to candidate equivalence classes. An extended set of *distance-1* simulation vectors are produced by flipping one bit at a time from C_{ex} [14].

While simplifying M_m in the $\&cec$ command, pairs of nodes from each class are represented in the temporary internal miter M_i and checked by the SAT solver S. All those nodes proved equivalent are merged in order to simplify M_m under verification and C_{ex} are used to disprove equivalences in further iterations. This iterative process of miter refinement and checking is executed until: *(i)* the miter is fully solved (M_m is proved $unsat$); *(ii)* a non-equivalent comparing point is detected (M_m is proved sat); or *(iii)* a resource limit is reached. We have observed that the most advantageous strategy to accelerate the ABC command $\&cec$ is to apply graph partitioning on M_m and M_i for enabling parallel SAT solving.

B. Graph Partitioning

The SAT problems arising from the proof of equivalence between pairs of POs or internal nodes are intrinsically independent of each other. These problems are encoded together in the same graph due to the natural logic sharing introduced during logic synthesis and optimization. We can exploit such an intrinsic independence of the problems to solve them in separate batches (partitions). Considering a miter represented as a multi-output AIG, we have three main motivations for performing the graph (miter) partitioning based on miter POs, as follows:

- The SAT problems for checking pairs of POs or internal nodes are independent from each other.
- We have empirically observed that adjacent outputs in M_m tend to present more shared logic than randomly selected groups of outputs. It is due to the fact that RTL elaborators, which translate word-level design description into bit-level circuit, place bit-level flops next to each other in M_m.
- The set of SAT problems formulated for checking equivalent classes during SAT sweeping are encoded as subsequent outputs in M_i.

Miter partitioning provides a trade-off between data sharing and data independence during equivalence checking. On one extreme side, we have many SAT problems encoded into a single miter, which can be incrementally solved at the same SAT solver by exploiting shared clauses (all together). On the other extreme side, we can apply graph partitioning to extract the TFI cones related to the pairs of outputs under comparison and check each pair using independent SAT calls (all separate). In order to achieve an equilibrium between data sharing and data independence, we are proposing to create relatively large partitions comprising a subset of SAT problems. The partitioning enables to solve the subset of SAT problems in parallel by independent SAT solver instances. Algorithm 1 presents a top-level view of the graph partitioning procedure.

Algorithm 1: Top-level view of graph partitioning

```
1  Function GraphPartitioning()
2      input: N (number of partitions), miterM ( AIG )
3      output: P ( partitions )
4      int S; // partition size
5      int R; // division reminder
6      if ( M.nPO ≥ N ) then
7          S = M.nPO / N;
8          R = M.nPO % N;
9      else
10         S = 1;
11         R = 0;
12         N = M.nPO ;
13     if ( R > 0 ) then
14         S++; // to treat remaining POs
15     AIG * P [ N ]; // partitions as an array of AIGs
16     int i = 0, j = 0;
17     for each po in M do
18         appendTFICone( po, M, P[ i ]);
19         j++;
20         if ( j == S) then
21             i++; // move to the next partition
22             j = 0; // reset the counter
23             // if all remaining POs were processed
24             if ( --R == 0 ) then
25                 S--; // go back to the original size
26     return P;
```

Initially, Algorithm 1 checks whether it is feasible to decompose the given miter M (AIG) into the desired number of partitions *N*, as shown in lines 6-12. This checking ensures that the graph partitioning will be consistent by assigning feasible values to the number of partitions *N* and partition size *S*. When the division of $M.nPO$ by N produces a remainder R, the algorithm distributes the set of remaining POs by placing one more PO to the first R partitions. This distribution contributes

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2019.2946254, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

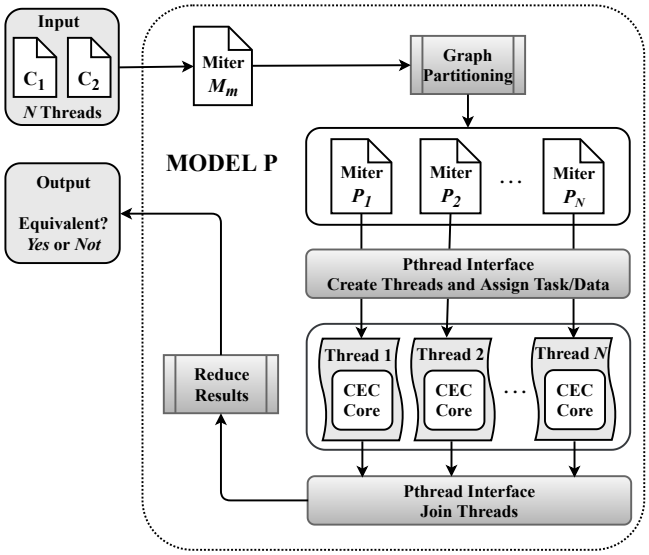


Fig. 3. Scheme of parallel CEC by main miter M_m partitioning.

to produce a workload balance among partitions.

In line 15 of Algorithm 1, all partitions start empty and the transitive fanin cones of subsequent POs are gradually appended to the current partition until the partition size is reached, represented by the loop in lines 17-25. The routine *appendTFICone* collects all the logic that a given PO depends on towards the PIs, recursively. In practice, *appendTFICone* executes a classical depth-first search (DFS) to collect/copy the necessary nodes to the current partition. Besides that, the proposed graph partitioning uses structural hashing to preserve logic sharing inside the partitions with some logic duplication among different partitions. However, since the CEC is a decision problem, logic duplication does not affect the solution quality. In other words, the CEC engine is not sensitive to logic duplication unlike other optimization problems, such as multi-level logic optimization and technology mapping. A TFI-based graph partitioning has been previously adopted in the context of graph propagation problems like simulation [47].

The proposed graph partitioning guarantees the soundness and completeness for CEC as follows. Let $P = \{P_1, P_2, ..., P_N\}$ be the set of N partitions of a given miter M. For each output pair $o_j \in M$ to be checked for equivalence, the proposed method guarantees that $o_j \in P_k$, being $0 < k \leq N$, so that each output pair belongs to some partition (*completeness*). Let TFI_{o_j} be the transitive fanin cone of o_j, the method also guarantees that the $TFI_{o_j} \subseteq P_k$ so that P_k contains all the logic needed to prove or disprove equivalence of o_j (*soundness*). The same graph partition can be applied for partitioning the main miter M_m as well as the internal miter M_i. The soundness and completeness properties hold for both cases since the M_i comprises smaller instances of the same problem represented in M_m.

C. Main Miter Partitioning

We are introducing the *Model P* in order to decompose M_m into a set of N independent partitions $P = \{P_1, P_2, ..., P_N\}$ and verify these sub-problems in parallel, as depicted in Fig. 3. Let C_1 and C_2 be the pair of circuits under verification. Initially, this model builds the miter M_m by comparing the corresponding pairs of outputs in multi-output AIG representation. Then, M_m is prepossessed by applying the graph partitioning introduced in Algorithm 1, leading to P.

Let T be the set of worker threads. The POSIX Threads (*Pthreads*) API is used for binding each $P_k \in P$ to a thread $T_k \in T$ (for $0 < k \leq N$). T_k is responsible for checking P_k by executing a completely independent instance of the ABC CEC core, illustrated in Fig. 2. Notice that in this approach the data independence is the key point, allowing to take advantage of many parallel runs of the sophisticated integration among *logic simulation*, *SAT sweeping* and *mitering*. Therefore, the *Model P* enables massive parallelism since each thread solves part of the problem without any dependencies and conflicts to other threads.

Finally, all threads in T are joined to the main thread and the intermediate solutions are combined to deliver the result of equivalence checking. If all the threads return the answer *equivalent*, then it means that the circuits under verification C_1 and C_2 are equivalent. In this case, the final runtime of CEC *Model P* is defined by the latest thread in T. However, when at least one pair of outputs is proved *non-equivalent*, the *Model P* enables earlier termination to the verification process since this model explores the solution space quickly. For instance, if a given thread T_k proves that at least one pair of outputs $o_j \in P_k$ is non-equivalent, then T_k can report the input/output patterns for debugging P_k and broadcast a stop signal to T.

The proposed approach lies closer to the middle of the spectrum between data sharing (all together) and data independence (all separate), enabling faster equivalence checking. This solution works like a divide-and-conquer approach to decompose a problem into smaller ones, enabling efficient parallel processing on multi-core platforms with shared memory. Moreover, since the partitions in P are completely independent to each other, the proposed approach can be easily extended to exploit the advantages of distributed computing in cloud platforms.

D. Internal Miter Partitioning

We are introducing the *Model S* in order to speed up the SAT sweeping applied for checking equivalences of internal nodes from M_m, as depicted in Fig. 4. At each iteration of the *Model S*, a temporary multi-output miter M_i is built to encode many SAT problems derived during SAT sweeping. Therefore, the same principle of graph partitioning presented in Algorithm 1 is applied to decompose M_i into a set of N' partitions $S = \{S_1, S_2, ..., S_{N'}\}$. After partitioning M_i, the *Pthreads* API is used to launch a set of threads T' so that each $T'_k \in T'$ is responsible for executing an independent instance of the SAT solver Minisat [18] to decide about the equivalences represented in S_k (for $0 < k \leq N'$).

Notice that the proposed approach interleaves serial and parallel sections of the code at each iteration, as shown in Fig. 4. Therefore, before starting the next iteration of the CEC *Model S* loop, all threads must finish and return partial results.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2019.2946254, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

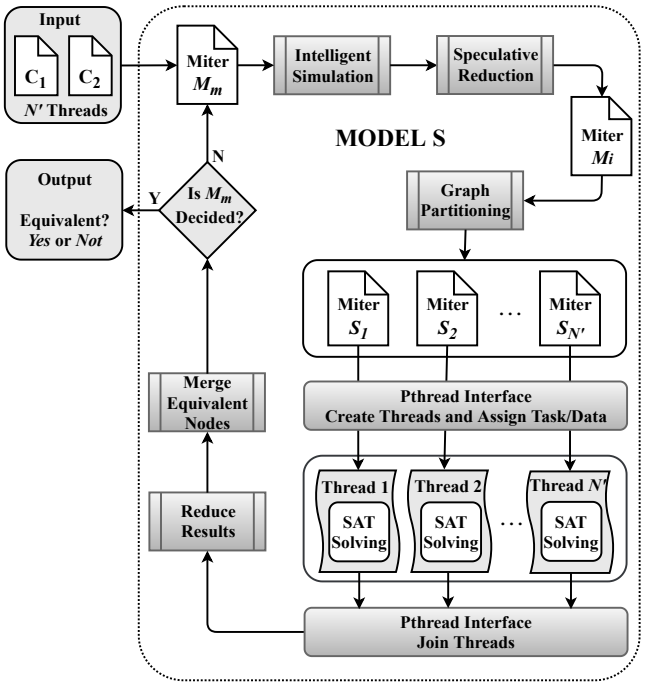


Fig. 4. Scheme of parallel CEC by internal miter M_i partitioning.

At each iteration, the pairs of nodes proved to be equivalent are collected to refine M_m. Moreover, a set of SAT counter-examples C_{ex} is collected from several SAT calls to produce simulation vectors for the next iteration. Auxiliary routines are used for reducing back all these partial results representing C_{ex} and equivalent nodes. This way, the runtime of the latest thread in T' defines the delay before moving to the next iteration of CEC *Model S*.

The amount of SAT problems encoded in M_i as well as the complexity of these problems are strongly dependent on the design structure. Typically, M_m comprises several complex SAT problems whereas M_i comprises a lot of small and easy SAT subproblems. Considering this scenario, it is a challenge to find a good trade-off between data sharing and data independence in M_i in order to perform a partitioning that leads to a good workload balance among the threads.

It is worth mentioning that many other applications which employ SAT sweeping can benefit from the proposed *Model S*. In optimization problems based on SAT sweeping, such as redundancy removal and signal correspondence [48], it is not wise to partition the input graph because optimization opportunities tend to be lost due to the negative bias of partition boundaries. Therefore, in such optimization problems, the proposed *Model P* is less useful. On the other hand, the proposed *Model S* can enable parallelism to solve the SAT sweeping problems encoded in the temporary miter M_i, without adversely impacting the quality of results of such optimization problems.

E. Combined Miter Partitioning

We have introduced the models P and S for speeding up equivalence checking by exploiting parallelism at different levels of the CEC engine. A strong feature of our approach is that both models can work together, cooperating to improve the CEC runtime. Since each partition P_k created in the *Model P* leads to an independent CEC subproblem, the proposed *Model S* can be directly applied to speed up SAT sweeping inside of P_k. It means that each thread $T_k \in T$, created in *Model P*, can create a set of subthreads T' in *Model S* to speed up the computation. The amount of threads working at each model can be specified by program switches followed by the number of threads, *i.e.,* -P N and -S N'. This way, one can fine tune the CEC engine to get the best thread workload balance based on the characteristics of the design under verification.

Since the amount of parallelism and work available in M_m and M_i are strongly related to the design structure, the ability to run *Model P* and *Model S* together helps to deal with a wide range of design characteristics. It has been confirmed in our experiments where the combined execution of models P and S achieves higher speedups, leading to the best results for some benchmarks. Moreover, when considering massively parallel environments available in cloud computing platforms, both models allow for a synergistic integration to fully exploit the power of distributed and shared memory computing. For instance, *Model P* can be used to partition the initial problem and so distribute tasks to many computing nodes in the cloud whereas *Model S* can exploit the potential of multi-core architectures at each node.

TABLE I
THE SET OF SIX CIRCUITS OBTAINED BY APPLYING THE ABC COMMAND *double* 10X, THE THREE MTM AIG NODES CIRCUITS FROM THE EPFL BENCHMARK SUITE [49] AND FOUR COMBINATIONAL CIRCUITS EXTRACTED FROM INDUSTRIAL DESIGNS.

			Original		*dc2-opt*	
Design	PI	PO	AND	Lev	AND	Lev
sin_10xd	24,576	25,600	5,545,984	225	0.93	0.73
arbiter_10xd	262,144	132,096	12,123,136	87	1.00	1.02
voter_10xd	1,025,024	1,024	14,088,192	70	0.63	0.94
square_10xd	65,536	131,072	18,927,616	250	0.90	0.99
sqrt_10xd	131,072	65,536	25,208,832	5,058	0.75	1.20
mult_10xd	131,072	131,072	27,711,488	274	0.90	0.96
sixteen	117	50	16,216,836	140	0.73	0.53
twenty	137	60	20,732,893	162	0.73	0.44
twentythree	153	68	23,339,737	176	0.72	0.44
circuit1	120K	18K	470K	300	0.84	0.15
circuit2	1.2M	1.8M	5.7M	24K	0.66	0.01
circuit3	1.5M	1.2M	6.0M	1K	0.72	0.25
circuit4	1.6M	1.2M	6.1M	9K	0.72	0.04

V. EXPERIMENTAL RESULTS

The proposed approaches have been implemented in C programming language using *Pthreads* and incorporated in the ABC command &*cec* [6]. The three ways of parallelizing CEC presented in Section IV can be enabled in the &*cec* command using the switches $-P$ and $-S$, together or separately, followed by the desired number of threads applied in each model.

A. Benchmark Circuits

Table I presents the set of benchmark circuits addressed in the experiments. These circuits can be divided into three different groups:

i) Since this work is focusing on speeding up CEC for large designs, we have selected the three benchmark circuits comprising more than ten million (MtM) AIG nodes from EPFL suite [49]. These circuits comprise AIGs with sixteen, twenty and twenty three million nodes, corresponding to random Boolean functions with complex implementation cost. These circuits are large enough to challenge CEC engines.

ii) Additionally, we have applied the ABC command *double* 10 times for six other circuits from the EPFL suite to generate large AIGs. This command doubles the AIG size by creating two copies placed side by side, each one with its own primary inputs and primary outputs. Although these circuits are synthetic, they are very similar to the combinational logic cloud extracted from a heavily pipelined industrial design, in which the pipeline stages are represented as different copies of the same design. Remembering that, in real designs, combinational logic clouds between pipeline stages are independently of each other and the CEC tools must to check all the internal combinational outputs as well.

iii) The last set of circuits comprises four industrial designs with up to 6 million AIG nodes, as shown in the bottom rows in Table I. These industrial designs comprise sequential elements (latches), being the smallest circuit comprising about a hundred thousand latches whereas the other three ones comprising about a million latches each one. Notice that the CEC tool must verify not only the POs but also all the internal combinational outputs that are connected to latches/flip-flops inputs. Therefore, since the sequential logic is not relevant for CEC, we are presenting only the combinational logic clouds of these circuits by treating latches as pseudo PIs/POs. These purely-combinational versions of the circuits have exactly the same number of combinational comparing points as in the original (sequential) versions of these circuits, producing exactly the same results in terms of CEC runtime.

In a typical scenario, CEC is used to check equivalence between original and optimized versions of the same design after logic synthesis and/or technology-dependent optimization. To reproduce this scenario, we have applied firtly the script *dc2* available in ABC to the designs shown in Table I. The script performs area-driven, delay-constrained, multi-level optimization process using *rewriting*, *balancing* and *refactoring* algorithms. The ratio in terms of AIG nodes and levels between the original and the optimized circuits is presented in the last two columns in Table I, *i.e., dc2-opt / Original*. In the following experiments, we have checked the equivalence between the original designs and the *dc2*-optimized ones, where all the pairs of circuits under verification are logically equivalent.

In the first experiments, the proposed parallel CEC has been compared to a commercial verification tool. In the sequence, we present a scalability analysis of the proposed models in a massive parallel environment when comparing to the reference single-threaded method available through the ABC command $\&cec$. Finally, we compare the proposed models to the parallel SAT solver Cube-and-Conquer [39] as well as we present a discussion regarding the related method EQUIPE [40].

TABLE II
RUNTIME COMPARISON AMONG THE COMMERCIAL TOOL AND THE PROPOSED MODELS RUNNING AT FOUR THREADS, IN (H:M:S).

Design	Commercial (4 threads)	-P 4	-S 4	-P 2 -S 2	Speedup
sin_10xd	1:02:09	**0:07:09**	0:11:15	0:08:29	8.68x
arbiter_10xd	0:51:53	**0:00:51**	0:01:36	0:01:06	60.56x
voter_10xd	**1:19:37**	4:19:08	4:54:53	4:09:05	0.32x
square_10xd	2:57:02	**0:03:28**	0:07:45	0:04:52	51.13x
sqrt_10xd	69:03:25	**3:31:50**	6:47:06	6:38:47	19.56x
mult_10xd	5:20:41	**0:09:04**	0:20:15	0:12:46	35.34x
sixteen	21:21:26	4:22:54	3:34:31	**1:45:06**	12.19x
twenty	38:08:40	3:35:22	6:27:38	**2:19:40**	16.39x
twentythree	49:46:06	3:56:46	8:11:20	**2:52:52**	17.27x
circuit1	0:08:22	0:14:30	**0:07:14**	0:13:58	1.16x
circuit2	20:47:20	**0:11:19**	0:15:15	0:12:09	110.28x
circuit3	15:06:45	**0:09:33**	0:09:38	0:09:42	95.01x
circuit4	17:49:01	1:07:01	1:00:52	**0:49:33**	21.58x
Average	**18:44:48**	**1:40:41**	**2:28:24**	**1:30:37**	**34.57x**

B. Comparison to Commercial Verification Tool

In this experiment, we are comparing the proposed models for parallel CEC to a commercial verification tool. The results were collected in a server with 64GB of shared RAM and an Intel®Core®i7-7700K CPU at 4.20GHz, where the processor has four physical cores. The tools under comparison were executed in a 64-bit Linux distribution and the runtimes were measured using the Linux bash command $time$ (real).

Both the commercial tool and the proposed models were executed using four threads in order to exploit the potential of the four physical cores available on the server. We have executed Model P and Model S separately and combined. Table II presents the absolute runtimes for each method in the format ($hours$: $minutes$: $seconds$). The last column in Table II presents the best speedups provided by the proposed approaches when comparing to the commercial tool. The experimental results have demonstrated that the proposed models for parallel CEC present significant smaller runtimes than the commercial verification tool when running at the same number of threads. For three out of four industrial designs, the proposed method presented great speedups when comparing to the commercial tool performance. Notice that, in many cases, the commercial tool took several hours or more than a day for verifying the designs, whereas the proposed approach took only few minutes/hours. On average, the proposed method is 34.57x faster than the commercial tool.

C. Scalability Analysis Comparing to ABC Command $\&cec$

In order to assess the scalability of the proposed parallel models to many threads, a set of experiments has been carried out using a server with 128GB of shared RAM and four Intel®Xeon®CPU E7- 4860 processors operating at 2.27GHz, where each processor has 10 physical cores, corresponding to the total of 40 cores. ABC tool was compiled using GNU g++ version 6.1.0 and executed in a 64-bit Linux distribution. The runtimes were measured using the Linux bash command $time$ (real).

Unfortunately, we do not have a commercial tool available on this server with greater computing power. Therefore, in the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2019.2946254, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

TABLE III
RUNTIME COMPARISON AMONG THE ORIGINAL ABC COMMAND *&cec* AND THE PROPOSED MODELS *P* AND *S* RUNNING SEPARATELY, IN (H:M:S).

Design	ABC *&cec*	*-P* 4	*-S* 4	*-P* 10	*-S* 10	*-P* 20	*-S* 20	*-P* 30	*-S* 30	*-P* 40	*-S* 40
sin_10xd	1:04:52	0:15:33	0:23:56	0:06:28	0:16:27	0:03:09	0:12:55	0:02:12	0:11:48	**0:02:05**	0:12:14
arbiter_10xd	0:05:17	0:01:35	0:03:36	**0:01:18**	0:03:18	0:01:38	0:03:16	0:02:08	0:03:22	0:02:41	0:03:28
voter_10xd	24:13:00	5:56:16	7:34:12	2:51:05	3:25:12	1:17:48	1:46:06	0:50:16	1:12:14	**0:39:12**	1:02:42
square_10xd	0:33:53	0:07:58	0:17:28	0:03:23	0:14:23	0:01:56	0:13:02	0:01:28	0:12:41	**0:01:16**	0:12:56
sqrt_10xd	21:34:04	7:45:45	14:17:51	4:05:54	12:57:21	2:18:04	12:19:09	1:34:57	12:09:34	**1:21:22**	12:10:35
mult_10xd	1:21:10	0:19:37	0:43:36	0:08:27	0:36:30	0:04:35	0:33:28	0:03:25	0:32:42	**0:03:03**	0:33:26
sixteen	8:27:33	7:07:47	7:17:48	1:43:48	7:49:08	0:29:03	7:08:11	0:15:21	7:10:25	**0:13:28**	7:24:37
twenty	14:35:59	4:14:36	14:53:13	3:48:59	13:39:38	0:27:43	14:32:43	0:15:27	13:55:20	**0:14:25**	13:50:52
twentythree	18:51:30	5:08:32	17:50:37	2:01:59	17:09:33	0:50:28	18:41:32	0:33:01	18:46:43	**0:17:56**	17:03:48
circuit1	0:12:57	0:28:57	0:14:38	0:04:59	0:15:42	0:01:31	0:15:11	0:00:52	0:16:06	**0:00:46**	0:15:56
circuit2	0:34:36	0:23:44	0:33:46	0:21:39	0:30:57	0:20:33	0:29:44	**0:19:35**	0:32:03	0:20:13	0:39:21
circuit3	0:28:11	0:21:54	0:21:53	0:17:16	0:20:36	0:12:57	0:21:36	**0:11:42**	0:23:13	0:11:54	0:24:39
circuit4	4:07:15	3:03:10	2:47:20	2:14:39	2:08:21	1:39:05	1:48:59	0:54:16	1:36:01	**0:53:21**	1:29:30
Average	**7:23:52**	**2:42:43**	**5:10:46**	**1:22:18**	**4:34:24**	**0:36:02**	**4:29:41**	**0:23:26**	**4:23:15**	**0:20:08**	**4:15:42**

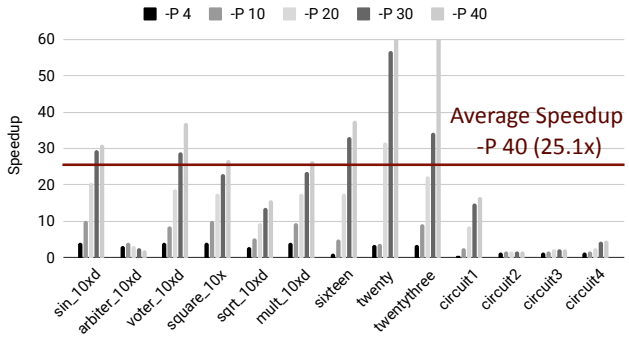


Fig. 5. Speedups by the main miter partitioning (*Model P*).

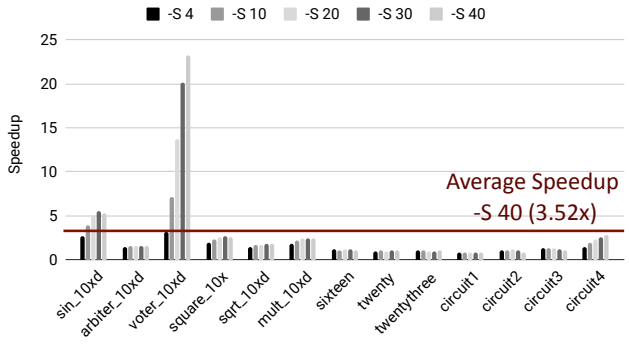


Fig. 6. Speedups by the internal miter partitioning (*Model S*).

following experiments, we are comparing the proposed parallel CEC approaches to the state-of-the-art serial CEC from ABC. In the first experiments, we compare the proposed models P and S separately. The goal of this experiment is to measure the speedup and the scalability that each model can bring to the verification process as the thread count increases. Table III presents the runtime for the original single-thread CEC and for parallel models P and S running at 4 up to 40 threads. The speedup introduced by each model according to the thread count is shown in Fig. 5 and in Fig. 6.

Regarding the benefits of the proposed approaches, *Model P* leads to more significant improvement than *Model S*. For most circuits, *Model P* works 30x faster than the original *&cec* command. When verifying the industrial case "circuit1", we have observed interesting speedups up to 16.71x faster than *&cec* performance. For the other three industrial circuits, we have profiled the CEC and observed that these cases have few comparing points (outputs) that are significantly more complex for verifying than all the others. Therefore, the overall runtime of *Model P* is dominated by the verification of these few outputs. Such a runtime behavior is not only related to the granularity of parallelism but it is also related to the complexity of the SAT problem. As can be seen in Table I, the last three industrial circuits were intensively optimized by the *dc2* script, leading to aggressive reduction in the number of nodes and mainly in the number of levels of these circuits. The optimization created very challenge CEC instances that are hard even for parallel SAT solvers which work in a much finer-grain level of parallelism than the proposed method, as presented in the next subsection.

Moreover, for the circuits "twenty" and "twentythree", we have observed super-linear speedups of 60.73x and 63.08x, respectively. In these cases, graph partitioning enabled a good balance between data sharing and data independence. Besides that, the CEC engine is based on incremental SAT solving by sharing clauses among successive SAT calls [14], [19]. Therefore, the miter partitioning can lead to a better incremental behavior in the SAT solving heuristics, since each thread is applied to a smaller set of problems using separate SAT solvers. In order to assess the influence of graph partitioning in the super-linear speedups, we have applied the proposed *Model P* to decompose the miters into 40 partitions and then solve each partition sequentially using the original version of the ABC command *&cec*. Table IV presents the absolute runtimes for the *&cec* command when verifying the MtM circuits serially with and without graph partitioning. Notice that experimental results support our hypothesis that the partitioning introduced a positive bias in the CEC runtime. On average, the *&cec* has processed the partitioned miters 3.11x faster than the conventional approach, contributing for

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2019.2946254, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

TABLE IV
IMPACT OF GRAPH PARTITIONING ON MITER SOLVING WITH SERIAL *&cec*, RUNTIME IN (H:M:S).

Design	ABC *&cec* No Partitioning	ABC *&cec* 40 Partitions	Speedup
sixteen	8:27:33	3:01:51	2.79x
twenty	14:35:59	4:29:26	3.25x
twentythree	18:51:30	5:45:13	3.28x
Average	**13:58:21**	**4:25:30**	**3.11x**

TABLE V
RUNTIME COMPARISON BETWEEN ORIGINAL ABC COMMAND *&cec* AND THE COMBINED MODELS *P* AND *S* RUNNING AT 40 THREADS, IN (H:M:S).

Design	ABC *&cec*	-P 4 -S 10	-P 10 -S 4	-P 20 -S 2
sin_10xd	1:04:52	0:03:55	0:02:32	**0:02:04**
arbiter_10xd	0:05:17	0:01:03	**0:00:47**	0:01:20
voter_10xd	24:13:00	0:39:45	**0:37:05**	0:37:26
square_10xd	0:33:53	0:03:21	0:01:54	**0:01:27**
sqrt_10xd	21:34:04	5:33:28	2:38:45	**1:44:48**
mult_10xd	1:21:10	0:09:09	0:04:51	**0:03:36**
sixteen	8:27:33	1:54:34	1:14:02	**0:28:07**
twenty	14:35:59	2:17:06	1:42:43	**0:27:01**
twentythree	18:51:30	2:21:32	2:22:09	**1:06:56**
circuit1	0:12:57	0:28:45	0:04:58	**0:01:34**
circuit2	0:34:36	0:22:13	0:20:30	**0:18:10**
circuit3	0:28:11	0:16:41	0:12:30	**0:11:52**
circuit4	4:07:15	1:17:01	**0:59:30**	1:09:58
Average	**7:23:52**	**1:11:26**	**0:47:52**	**0:28:48**

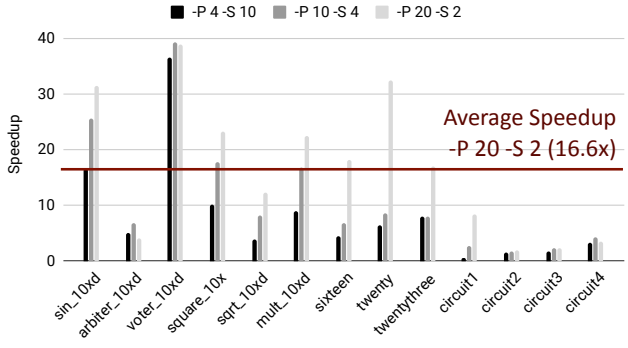


Fig. 7. Speedups by combining models *P* and *S* at 40 threads.

the supper-linear speedups.

It is harder to get high speedups in *Model S* because many smaller SAT problems are created in the interleaved sections of sequential and parallel codes in the CEC core. Actually, the SAT problem size and complexity depends on the characteristics of the designs under verification. Therefore, one should not discard *Model S* since it can be advantageous for certain designs, and it can also be combined to work together with *Model P*.

By combining models P and S, we enable extra opportunities to trade-off data sharing and data independence in the CEC engine. In this experiment, we consider three different thread configurations for combining both models using the total of 40 threads. In the first configuration ($-P$ 4 $-S$ 10), we are considering fewer partitions of the main miter and more partitions of the internal miter. In the second one ($-P$ 10 $-S$ 4), we have swapped the thread count between the main miter and the internal miter partitionings. In the last configuration ($-P$ 20 $-S$ 2), we have increased the thread count for the main miter partitioning since it has presented the best results in previous experiments. The absolute runtime values of each configuration are presented in Table V and the respective speedups are shown in Fig. 7.

To demonstrate the advantages of combining both models, consider the results for the designs "voter_10xd" and "arbiter_10xd" shown in Fig. 7. The configuration ($-P$ 10 $-S$ 4) resulted in extra speedups for both designs when comparing to the models P and S running independently. For "arbiter_10xd", we have observed the speedup of 6.7x whereas in the previous experiments the speedups were 4x for *Model P* and 1.5x for *Model S*. Moreover, for "voter_10xd" circuit, we have observed the speedup of 39x which is practically linear (optimal) in terms of the number of threads. In the previous experiments, the speedups for "voter_10xd" were 37x and 23x for models P and S, respectively.

D. Comparing to Parallel SAT Solving

In this experiment, we created miters in the CNF format for all the pairs of circuits presented in Table I by applying the ABC script: "*&r file1.aig ; &miter file2.aig ; &write_cnf miter.cnf*". Then, we used the state-of-the-art parallel SAT solver Cube-and-Conquer (CnC) [39] to solve these CNF formulas. The CnC was obtained from [50], compiled and executed in the same platform with 40 cores used in the previous experiment. We set the CnC to run with 40 threads by editing the number of thread in the provided script "cube-lingeling.sh". We set a timeout of 24 hours for solving each miter. The CnC reached the timeout for all the miters and was not able to finish the splitting (divide) phase which is based on a variant of the lookahead SAT solver MARCH [51]. This modified lookahead solver is used as spliting tool for decomposing the problem based on cubes and then to solve the sub-problems in parallel (conquer) with an extended version of the LINGELING CDCL solver [33]. However, in this experiment, we have observed that the conquer phase did not start after 24 hours for all the miters.

We are completely sure that the CnC is a powerful technique for parallel SAT solving that provide great speedups for several complex applications. However, as discussed in [52], lookahead methods are interesting to solve small hard problems that require sophisticated heuristics, presenting limited scalability when comparing to CDCL solver for industrial instances. The pairs of circuits under verification, including the industrial designs, comprise millions of AIG nodes, leading to challenge instances of CEC for which the lookahead solver became a bottleneck. In this context, a lightweight heuristic for partitioning, such as the proposed one, is more scalable for large instances because it is based on intrinsic properties of the target problem.

As future extension of the proposed approach for parallel CEC, we suggest the integration of a parallel SAT solver to add one more level of parallelism in our solution. For

instance, one can apply the proposed *Models P* and *Model S* for miter partitioning and then to solve in parallel (level 1) the SAT problems created inside of each partition by executing independent instances of a parallel (level 2) SAT solver based on divide-and-conquer or portfolio techniques. Notice that this solution allows to keep taking advantage of the integration among logic simulation, SAT sweeping and mitering, however, replacing the serial SAT solver by a parallel one.

E. Discussion on EQUIPE Method

Regarding the previous parallel CEC method called EQUIPE [40], it is quite difficult to perform a direct comparison to such a method. At first, the methods are running on different platforms with the technology gap of almost a decade. Besides, it is not clear whether the authors are comparing against cec or $\&cec$ command so that $\&cec$ is significantly faster than cec. Moreover, it is hard to ensure that we are using exactly the same pairs of original and optimized designs as input to CEC. Therefore, in order to avoid an unfair comparison, we are presenting an analysis based on our solution and on the general information presented in [40].

The results produced by the EQUIPE method were collected on a platform containing a CUDA-enabled 8800GT GPU with 14 multiprocessors operating at 600 MHz and a CPU Intel Core 2 Quad operating at 2.4 GHz. The authors reported average speedups of one order of magnitude when comparing to a commercial tool and only up to 3.22x speedup when comparing to ABC tool. In the latter case, even using the 4 CPU cores and the 14 auxiliary GPU cores, the method was not able to speedup CEC beyond 3.22x. It is worth to notice that the CPU and GPU cores work at different frequencies and there is an additional cost related to the communication between CPU and GPU. Moreover, the authors mention that in some cases the internal miters need to be reconstructed, leading to runtime degradation, as shown in [40]. On the other hand, the parallel models P and S proposed in this work are able to verify designs with millions of AIG nodes. It scales to many cores and achieves significant improvement when comparing to $\&cec$ command, the latest CEC engine in ABC tool. Therefore, the proposed approach is more promising than the EQUIPE method when it comes for improving CEC for large designs.

F. Final Considerations and Applications

Table VI presents a summary of the best results and the respective thread configuration. In general, the configurations $-P$ 40 and $-P$ 10 $-S$ 4 lead to the best results. Notice that the proposed approach can significantly reduce the runtime of CEC when comparing to the single-thread method. One great improvement has taken place when the CEC runtime went down from 24h13min to only 37min and from 18h51min to only 18min. Overall, the proposed solutions have the potential to improve existing EDA environments.

Given that CEC is used as a building block in many computations, the parallel CEC engine proposed in this work can speed up other important tasks which depends on proving logic equivalence. Moreover, with additional customization that performs proper handling of fanout nodes during equivalence checking the scalable CEC techniques are also applicable in:

- SAT sweeping under observability *don't-cares* [53];
- node minimization with satisfiability, observability and external *don't-cares* [54];
- high-effort resynthesis for circuit delay, area, power dissipation and wiring congestion reduction using Boolean resubstitution [48];
- various traversal-based computations comparing functions of the nodes in terms of primary inputs under observability conditions (ATPG, redundancy removal, false path detection and removal), and others.

TABLE VI
SUMMARY OF THE BEST RESULTS AND CONFIGURATIONS FOR EACH DESIGN, IN (H:M:S).

Design	ABC $\&cec$	Parallel	Speedup	Config.
sin_10xd	1:04:52	0:02:04	31.33x	-P 20 -S 2
arbiter_10xd	0:05:17	0:00:47	6.76x	-P 10 -S 4
voter_10xd	24:13:00	0:37:05	39.17x	-P 10 -S 4
square_10xd	0:33:53	0:01:16	26.72x	-P 40
sqrt_10xd	21:34:04	1:21:22	15.90x	-P 40
mult_10xd	1:21:10	0:03:03	26.59x	-P 40
sixteen	8:27:33	0:13:28	37.67x	-P 40
twenty	14:35:59	0:14:25	60.73x	-P 40
twentythree	18:51:30	0:17:56	63.08x	-P 40
circuit1	0:12:57	0:00:46	16.71x	-P 40
circuit2	0:34:36	0:18:10	1.91x	-P 20 -S 2
circuit3	0:28:11	0:11:42	2.41x	-P 30
circuit4	4:07:15	0:53:21	4.63x	-P 40
Average	**7:23:52**	**0:19:39**	**25.66x**	

VI. CONCLUSIONS

The paper proposes a novel approach comprising three different models to enable parallelism and to speedup modern CEC engine. The models trade off data sharing and data independence by applying graph partitioning during mitering and SAT sweeping. Experiments lead to promising results in speeding up the verification task for large designs. In several cases, where the ABC or the commercial verification tool took more than one day for checking designs, the proposed approach has finished this task in a few minutes/hours. The proposed solution has additional practical benefits, in particular, the potential to improve the runtime and scalability of other applications in current EDA environments. Moreover, the proposed models can exploit massively parallel environments of cloud computing.

ACKNOWLEDGMENT

This work was partially supported by the Brazilian funding agencies CNPq and CAPES, and by SRC contracts 2710.001 and 2867.001. The authors would like to thank Prof. Keshav Pingali from the UT Austin for the courtesy to collect experimental results in the multi-processor servers of his research group during the Ph.D. research of the author Vinicius Possani.

REFERENCES

[1] R. K. Brayton, G. D. Hachtel, and A. L. Sangiovanni-Vincentelli, "Multilevel logic synthesis," *in Proc. of the IEEE*, vol. 78, no. 2, pp. 264–300, Feb. 1990.

[2] S. Chatterjee, A. Mishchenko, R. K. Brayton, X. Wang, and T. Kam, "Reducing structural bias in technology mapping," *in IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 12, pp. 2894–2903, Dec. 2006.

[3] P. Bjesse and K. Claessen, "SAT-based verification without state space traversal," in *the Proc. of Formal Methods in Computer Aided Design - FMCAD*, 2000, pp. 372–389.

[4] A. Mishchenko, M. Case, R. Brayton, and S. Jang, "Scalable and scalably-verifiable sequential synthesis," in *the Proc. of Int'l Conf. on Computer Aided Design - ICCAD*, 2008, pp. 234–241.

[5] S. Krishnaswamy, H. Ren, N. Modi, and R. Puri, "DeltaSyn: An efficient logic difference optimizer for ECO synthesis," in *the Proc. of Int'l Conf. on Computer Aided Design - ICCAD*, 2009, pp. 789–796.

[6] Berkeley Logic Synthesis and Verification Group. *ABC: A System for Sequential Synthesis and Verification*. http://www-cad.eecs.berkeley.edu/ alanmi/abc.

[7] L. Stok, "Developing parallel EDA tools [the last byte]," *in IEEE Design & Test*, vol. 30, no. 1, pp. 65–66, 2013.

[8] ——, "The next 25 years in EDA: A cloudy future?" *in IEEE Design & Test*, vol. 31, no. 2, 2014.

[9] Y. Matsunaga, "An efficient equivalence checker for combinational circuits," in *the Proc. of Design Automation Conference - DAC*, 1996, pp. 629–634.

[10] J. Moondanos, C. Seger, Z. Hanna, and D. Kaiss, "CLEVER: Divide and conquer combinational logic equivalence verification with false negative elimination," in *the Proc. of Int'l Conf. on Computer Aided Verificatio - CAV*, 2001, pp. 131–143.

[11] I.-H. Moon and C. Pixley, "Non-miter-based combinational equivalence checking by comparing BDDs with different variable orders," in *the Proc. of Formal Methods in Computer Aided Design - FMCAD*, 2004, pp. 144–158.

[12] A. Kuehlmann, V. Paruthi, F. Krohm, and M. K. Ganai, "Robust boolean reasoning for equivalence checking and functional property verification," *in IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 12, pp. 1377–1394, Dec. 2002.

[13] F. Lu, L.-C. Wang, K.-T. Cheng, and R. C.-Y. Huang, "A circuit SAT solver with signal correlation guided learning," in *the Proc. of Design, Automation and Test in Europe - DATE*, 2003, pp. 892–897.

[14] A. Mishchenko, S. Chatterjee, R. Brayton, and N. Een, "Improvements to combinational equivalence checking," in *the Proc. of Int'l Conf. on Computer Aided Design - ICCAD*, 2006, pp. 836–843.

[15] A. Mishchenko and R. K. Brayton, "Scalable logic synthesis using a simple circuit structure," in *the Tech. Dig. of Int'l Workshop on Logic & Synthesis - IWLS*, 2006.

[16] M. J. Heule and H. van Maaren, *Look-Ahead Based SAT Solvers*. Amsterdam, The Netherlands, The Netherlands: IOS Press, 2009, vol. 185, p. 155184.

[17] J. P. Marques-Silva, I. Lynce, and S. Malik, *Conflict-Driven Clause Learning SAT Solvers*. Amsterdam, The Netherlands, The Netherlands: IOS Press, 2009, vol. 185, pp. 131–153.

[18] N. En and N. Srensson, "An Extensible SAT-solver," in *the Proc. of Int'l Conf. on Theory and Applications of Satisfiability Testing*. Springer, Berlin, Heidelberg, 2003, pp. 502–518.

[19] ——, "Temporal induction by incremental sat solving," *in Electronic Notes in Theoretical Computer Science*, vol. 89, no. 4, pp. 543 – 560, 2003.

[20] D. Brand, "Verification of large synthesized designs," in *the Proc. of Int'l Conf. on Computer Aided Design - ICCAD*, 1993, p. 534537.

[21] G. S. Tseitin, *On the complexity of derivation in propositional calculus, Automation of reasoning*. Springer Berlin Heidelberg, 1983.

[22] A. Kuehlmann, "Dynamic transition relation simplification for bounded property checking," in *the Proc. of Int'l Conf. on Computer Aided Design - ICCAD*, 2004, pp. 50–57.

[23] R. Bryant, "Graph-based algorithms for boolean function manipulation," *in IEEE Trans. on Computers*, vol. C-35, no. 8, pp. 677–691, Aug. 1986.

[24] E. M. Clarke, O. Grumberg, and D. Peleg, *Model Checking*. MIT Press, 1999.

[25] C. Baier and J.-P. Katoen, *Principles of Model Checking*. MIT Press, 2008.

[26] A. Biere, C. Artho, and V. Schuppan, "Liveness checking as safety checking," *Electronic Notes in Theoretical Computer Science*, vol. 66, no. 2, pp. 160 – 177, 2002.

[27] R. P. Kurshan, *Verification Technology Transfer*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 46–64.

[28] M. R. Prasad, A. Biere, and A. Gupta, "A survey of recent advances in sat-based formal verification," *in Int. J. Softw. Tools Technol. Transf.*, vol. 7, no. 2, pp. 156–173, Apr. 2005.

[29] A. Biere and D. Kröning, *SAT-Based Model Checking*. Cham: Springer International Publishing, 2018, pp. 277–303.

[30] T. Balyo and C. Sinz, *Parallel Satisfiability*. Springer International Publishing, 2018, pp. 31–59.

[31] O. Roussel, "Description of ppfolio," in *the Proc. of SAT Challenge*, 2012, p. 46.

[32] Y. Hamadi, S. Jabbour, and L. Sais, "ManySAT: a parallel SAT solver," *in Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, vol. 6, 2009.

[33] A. Biere, "Lingeling, Plingeling, PicoSAT and PrecoSAT at SAT Race 2010," Technical Report 10/1, FMV Reports Series, Institute for Formal Models and Verification, Johannes Kepler University, Tech. Rep., 2010.

[34] T. Balyo, P. Sanders, and C. Sinz, "Hordesat: A massively parallel portfolio sat solver," in *the Proc. of Theory and Applications of Satisfiability Testing - SAT*, M. Heule and S. Weaver, Eds. Springer International Publishing, 2015, pp. 156–172.

[35] L. Amarú, P. Gaillardon, A. Mishchenko, M. Ciesielski, and G. D. Micheli, "Exploiting Circuit Duality to Speed up SAT," in *the Proc. of IEEE Computer Society Annual Symposium on VLSI*, July 2015, pp. 101–106.

[36] A. E. J. Hyvärinen, T. Junttila, and I. Niemelä, "A distribution method for solving sat in grids," in *the Proc. of Int'l Conf. on Theory and Applications of Satisfiability Testing*, ser. SAT'06, 2006, pp. 430–435.

[37] B. Jurkowiak, C. M. Li, and G. Utard, "A Parallelization Scheme Based on Work Stealing for a Class of SAT Solvers," *in Journal of Automated Reasoning*, vol. 34, no. 1, pp. 73–101, Jan. 2005.

[38] L. S. Youssef Hamadi, *Handbook of Parallel Constraint Reasoning*. Springer, 2018.

[39] M. J. H. Heule, O. Kullmann, S. Wieringa, and A. Biere, "Cube and Conquer: Guiding CDCL SAT Solvers by Lookaheads," in *the Proc. of Int'l Haifa Verification Conference - Hardware and Software: Verification and Testing*, K. Eder, J. Lourenço, and O. Shehory, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 50–65.

[40] D. Chatterjee and V. Bertacco, "EQUIPE: Parallel equivalence checking with GP-GPUs," in *the Proc. of Int'l Conf. on Computer Design - ICCD*, 2010, pp. 486–493.

[41] O. Wienand, M. Wedler, D. Stoffel, W. Kunz, and G.-M. Greuel, "An algebraic approach for proving data correctness in arithmetic data paths," in *the Proc. of Int'l Conf. on Computer Aided Verification*, ser. CAV '08, 2008, pp. 473–486.

[42] J. Lv, P. Kalla, and F. Enescu, "Efficient grbner basis reductions for formal verification of galois field multipliers," in *the Proc. of Design, Automation and Test in Europe - DATE*, March 2012, pp. 899–904.

[43] M. Ciesielski, C. Yu, W. Brown, D. Liu, and A. Rossi, "Verification of gate-level arithmetic circuits by function extraction," in *the Proc. of Design Automation Conference - DAC*, June 2015, pp. 1–6.

[44] F. Farahmandi and B. Alizadeh, "Groebner basis based formal verification of large arithmetic circuits using gaussian elimination and cone-based polynomial extraction," *in Microprocess. Microsyst.*, vol. 39, no. 2, pp. 83–96, Mar. 2015.

[45] A. Sayed-Ahmed, D. Groe, U. Khne, M. Soeken, and R. Drechsler, "Formal verification of integer multipliers by combining grbner basis with logic reduction," in *the Proc. of Design, Automation and Test in Europe - DATE*, 2016, pp. 1048–1053.

[46] A. Mishchenko, S. Chatterjee, R. Jiang, and R. K. Brayton, "FRAIGs: A unifying representation for logic synthesis and verification," UC Berkeley, Tech. Rep., 2005.

[47] D. Chatterjee, A. DeOrio, and V. Bertacco, "GCS: High-performance gate-level simulation with GPGPUs," in *the Proc. of Design, Automation and Test in Europe - DATE*, 2009, pp. 1332–1337.

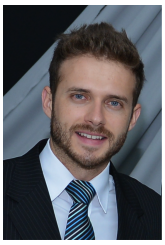
[48] A. Mishchenko, R. Brayton, J.-H. R. Jiang, and S. Jang, "Scalable don't-care-based logic optimization and resynthesis," *in ACM Trans. on Reconfigurable Technology and Systems*, vol. 4, no. 4, pp. 34:1–34:23, Dec. 2011.

[49] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "The EPFL combinational benchmark suite," in *the Tech. Dig. of Int'l Workshop on Logic & Synthesis - IWLS*, 2015.

[50] Marijn J.H. Heule. Cube-and-Conquer SAT solver. https://github.com/marijnheule/CnC.

[51] S. Mijnders, B. de Wilde, and M. Heule, *Symbiosis of Search and Heuristics for Random 3-SAT*. LaSh, 2010.

[52] M. J. H. Heule, O. Kullmann, and A. Biere, *Cube-and-Conquer for Satisfiability*. Springer International Publishing, 2018, pp. 31–59.
[53] Q. Zhu, N. Kitchen, A. Kuehlmann, and A. L. Sangiovanni-Vincentelli, "SAT sweeping with local observability don't-cares," in *the Proc. of Design Automation Conference - DAC*, 2006, pp. 229–234.
[54] A. Mishchenko and R. Brayton, "SAT-based complete don't-care computation for network optimization," in *the Proc. of Design, Automation and Test in Europe - DATE*, 2005, pp. 412–417.



Vinicius N. Possani (S'13) received the Bachelor's Degree and the Master's Degree in Computer Science from Federal University of Pelotas (UFPEL), Pelotas, Brazil, in 2013 and 2015, respectively. From April to August of 2017 he was a Ph.D. Visiting Research Scholar at ISS research group from University of Texas at Austin. He received the Ph.D. degree in Computer Science from Federal University of Rio Grande do Sul (UFRGS), Porto Alegre, Brazil, in 2019. His research interests include logic synthesis, technology mapping, verification and parallel computing applied to electronic design automation (EDA).

Alan Mishchenko graduated from Moscow Institute of Physics and Technology (Moscow, Russia) in 1993 with MS and received his PhD from the Glushkov Institute of Cybernetics (Kiev, Ukraine) in 1997. From 1998 to 2002 he was an Intel-sponsored visiting scientist at Portland State University. Since 2002, he has been a professional researcher in the EECS Department at UC Berkeley. Dr. Mishchenko shared the D.O. Pederson TCAD Best Paper Award in 2008 and the SRC Technical Excellence Award in 2011 for work on ABC. His research interests are in developing computationally efficient methods for synthesis and verification.

Renato P. Ribas (M'12) received the B.S. degree in Electrical Engineering from Federal University of Rio Grande do Sul (UFRGS), Porto Alegre, Brazil, in 1991, the M.S. degree in Electrical Engineering from University of Campinas (Unicamp), Campinas, Brazil, in 1994, and the Ph.D. degree in Microelectronics from Institut National Polytechnique de Grenoble, France, in 1998. He was visiting research at University of British Columbia, Vancouver, Canada, in 2010-2011. He is currently professor in the Department of Applied Informatics, at Institute of Informatics, UFRGS, since 2000. His research interests are digital IC design and CAD development.

Andre I. Reis (M'99-SM'05) received the B.S. degree in Electrical Engineering from Federal University of Rio Grande do Sul (UFRGS), Porto Alegre, Brazil, in 1991, the M.S. degree in Computer Science also from UFRGS, in 1993, and the Ph.D. degree in Automatic and Microelectronics Systems from UMII, Montpellier, France, in 1998. He is currently professor in the Department of Applied Informatics, at Institute of Informatics, UFRGS, since 2000. He was a visiting researcher at University of Minnesota, Minneapolis, USA, in 2004-2005.