Simulation-based Parallel Sweeping: A New Perspective on Combinational Equivalence Checking

Tianji Liu and Evangeline F.Y. Young
Department of CSE, CUHK

THE CHIPS TO SYSTEMS CONFERENCE
62
香港中文大學
The Chinese University of Hong Kong
SPONSORED BY
CEDA IEEE Council on Electronic Design Automation
sigda acm

Outline

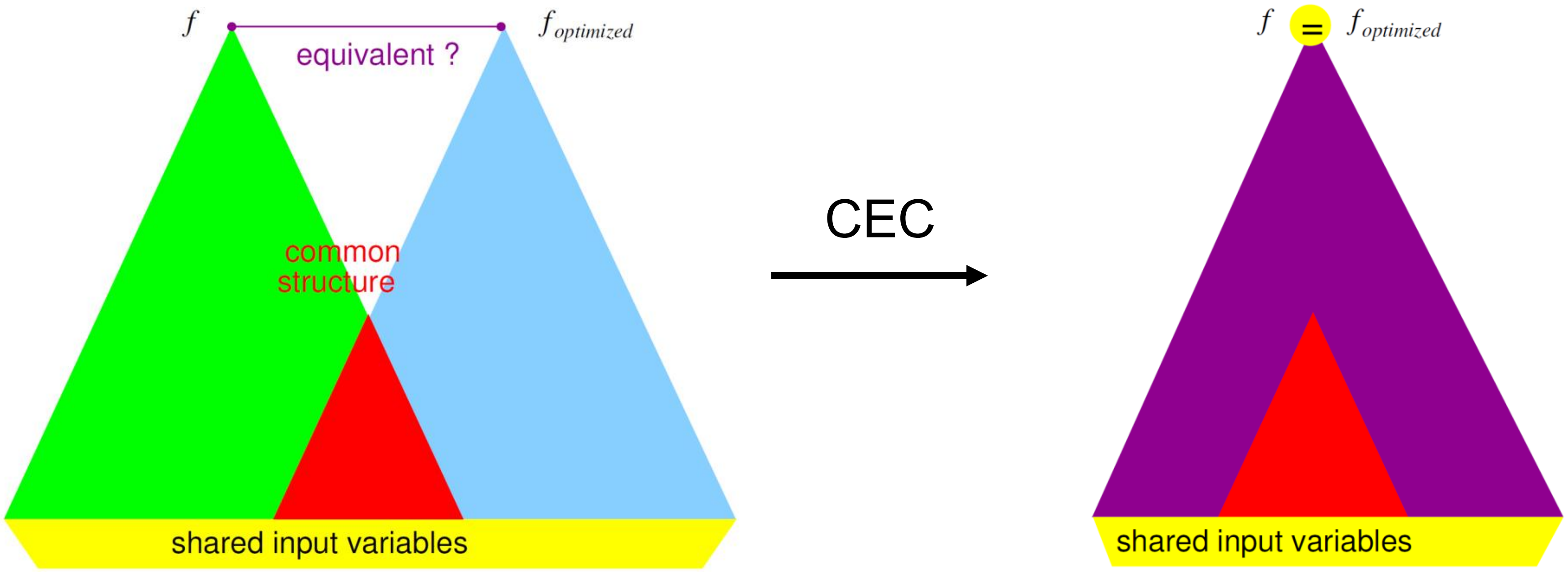
- Background & Motivation
- Simulation-based CEC Engine
  - Massively Parallel Exhaustive Simulation
  - Local Function Checking
  - Overall Flow
- Experimental Results
- Summary

Background & Motivation

THE CHIPS TO SYSTEMS CONFERENCE

Combinational Equivalence Checking (CEC)

- Prove the equivalence of two netlists implementing the same circuit
- CEC is **co-NP-complete**: no universally efficient algorithm
- Applications: verification, logic synthesis, functional ECO



Source: Armin Biere, “SAT in Formal Hardware Verification”, SAT’05 talk

CEC by Sweeping

- Miter: sharing PIs of the two circuits, combine POs with XORs
  - Two circuits are equivalent <=> miter is const-0

- Assumption: there are **internal equivalent nodes** in the miter
  - Usually true if one circuit is the optimized version of another (i.e., during synthesis)

- Sweeping
  - Gradually reduce the miter by proving and merging internal equivalent nodes

  → Easier to prove the POs
  - Use a formal method (e.g., BDD, SAT) to perform the proof

CEC Solutions: Past & Present

- Early years: binary decision diagrams (BDDs)
  - Not scalable to large circuits
- Recent years: Boolean satisfiability (SAT)
  - Huge improvements of SAT solving performance (clause learning, inprocessing, …)
- SAT is not always optimal for CEC (due to co-NP-completeness)
  - Example: computer algebra methods for arithmetic circuits

- Conclusion: **new solutions of CEC are constantly needed**

"The Bitter Lesson" from AI Community

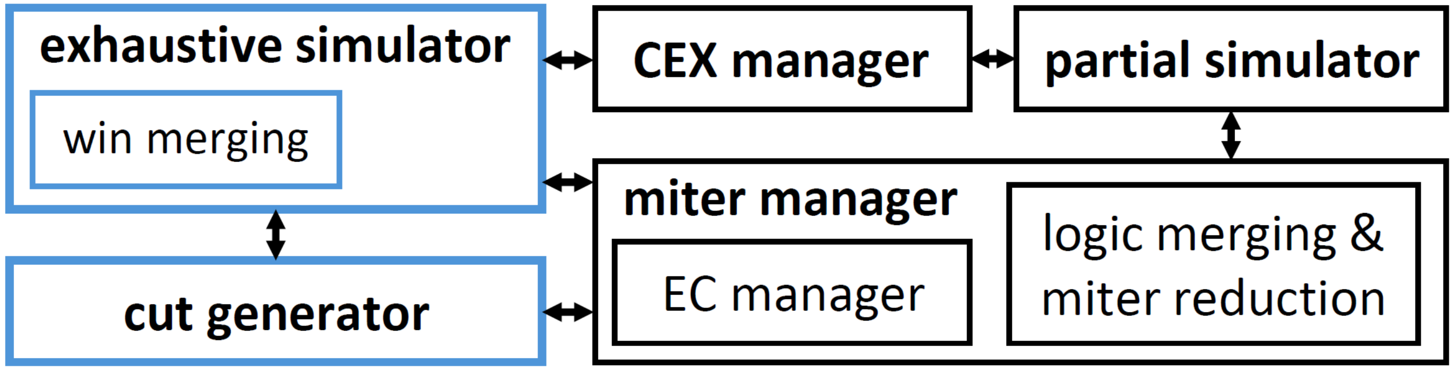
- Richard Sutton (2024 Turing Award recipient):
  - "The biggest lesson that can be read from 70 years of AI research is that general methods that leverage computation are ultimately the most effective."
  - Explanation: the methods that are <u>more general</u> and <u>exploit massive computational power</u> usually outperform those relying on domain knowledge.

- Will there be a similar case in CEC?

Overview of Our CEC Approach

- General method: use **exhaustive simulation** to prove equivalences
- Massive computational power: parallel computation exploiting **GPUs**
- A **local function checking** scheme for reducing time complexity

NVIDIA
CUDA



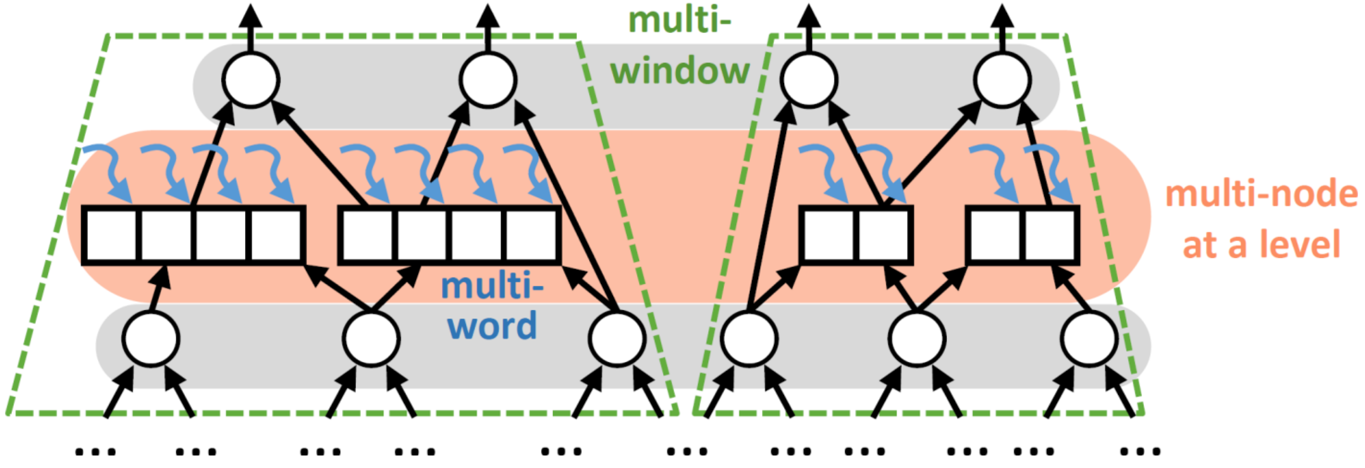
Massively Parallel Exhaustive Simulation

THE CHIPS TO SYSTEMS CONFERENCE

SPONSORED BY CEDA sigda

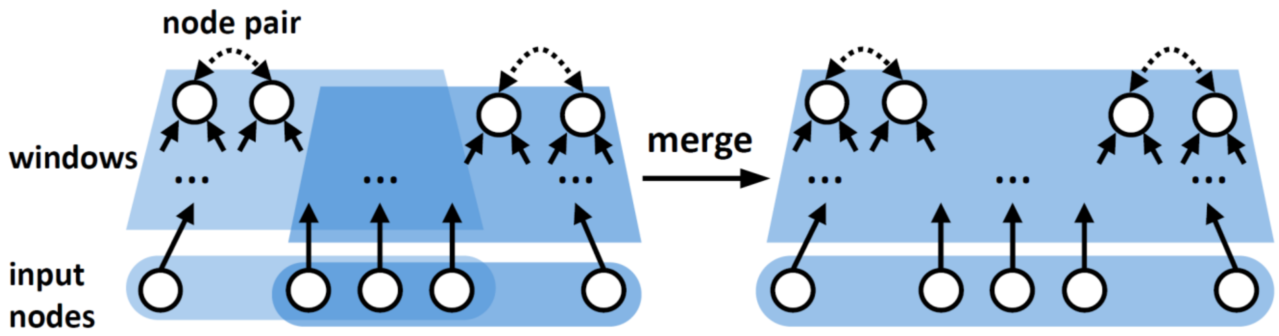
Exhaustive Simulation

- Objective of Exhaustive Simulator
  - Check equivalence of many candidate node pairs
  - For each pair, enumerate all possible patterns at inputs, compute response at the two nodes
- Three dimensions of parallelism
  1. Parallel simulation of different pairs (**windows**)
  2. Level-wise parallel node simulation
  3. Parallel multi-word simulation for a node



Window Merging

- Merge windows with high overlaps
  - One window for simulating multiple pairs
  - Reduce simulation effort and #windows



- Approach
  - Sort the windows in lexicographical order of input nodes
  - Cluster neighboring windows into one, with a limit on max #inputs
  - Example: inputs {a, b}, {a, b}, {a, b, c}, {a, e}, {a, f} -> {a, b, c}, {a, e, f} (max #inputs = 3)

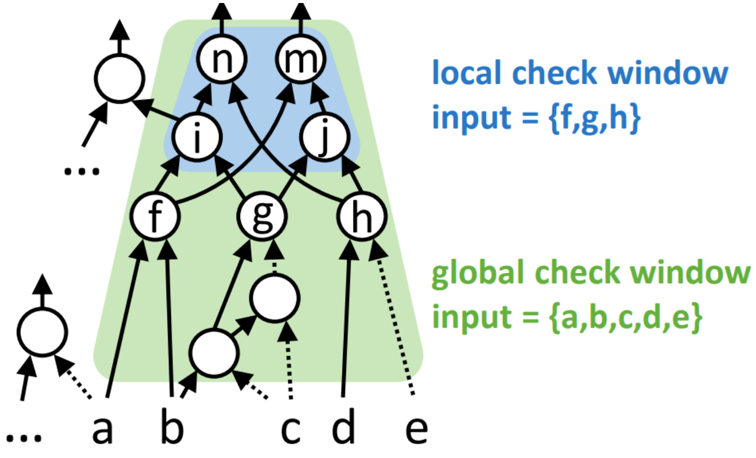
Local Function Checking

THE CHIPS TO SYSTEMS CONFERENCE

SPONSORED BY

Local Function Checking

- Basic idea
  - Check the equivalence of two nodes' functions in terms of a common cut (input = cut leaves)
  - Restrict the complexity of exhaustive simulation (exponential in input size)
- Properties
  - EQ local function ⇒ EQ pair
  - NEQ local function ⇏ NEQ pair
  - NEQ local function, EQ pair ⇒ patterns leading to NEQ are **satisfiability don't cares** (SDCs)
- Main objective: avoid SDCs at the cut nodes; how?
  - Check multiple cuts for a node pair
  - Ensure the "quality" of the cuts



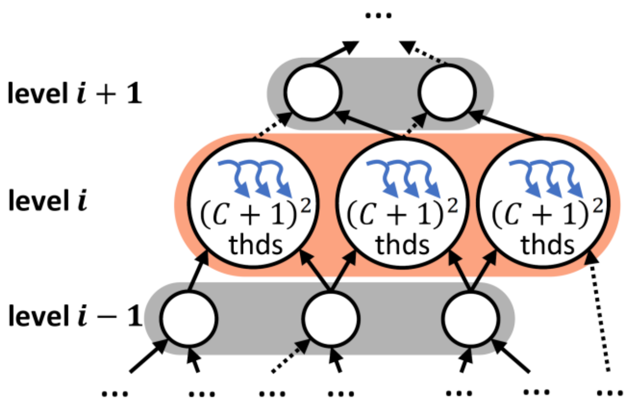
Cut Generation for Node Pairs

- Cut enumeration: generate C cuts $P(n)$ (size $< k$) for every <u>single</u> node

$$E(n) = \left\{u \cup v \;:\; u \in P(n_0) \cup \left\{\{n_0\}\right\}, v \in P(n_1) \cup \left\{\{n_1\}\right\}, |u \cup v| \leq k\right\}$$
$$P(n) = \text{the best } C \text{ cuts in } E(n)$$

- Level-wise parallel computation, using the method of [1]
- Generate <u>common cuts</u> for a node pair n, m

$$P(n,m) = \{u \cup v \;:\; u \in P(n), v \in P(m), |u \cup v| \leq k\}$$



Source: [1]

[1] Tianji Liu, Lei Chen, Xing Li, Mingxuan Yuan, Evangeline F.Y. Young, "FineMap: A Fine-grained GPU-parallel LUT Mapping Engine", ASP-DAC, 2024.

Ensuring Cut Quality

- How to select the best C cuts from many cuts?

Pass	Main Metric	Tie-breaker Metric 1	Tie-breaker Metric 2
1	large avg. fanout	small cut size	small avg. level
2	small avg. level	small cut size	large avg. fanout
3	large avg. level	small cut size	large avg. fanout

- Multiple passes with different selection metrics to ensure high diversity of cuts
- High usability of common cuts: encourage similar cuts of two nodes in a pair

Local Function Checking Approach

- In short: on-the-fly exhaustive simulation along with the cut generation process
  - A constant-sized common cut buffer
  - Collect common cuts into the buffer by level-wise parallel cut enumeration
  - Once the buffer is full, launch a batch of exhaustive simulation
  - Clear the buffer and continue the cut enumeration process

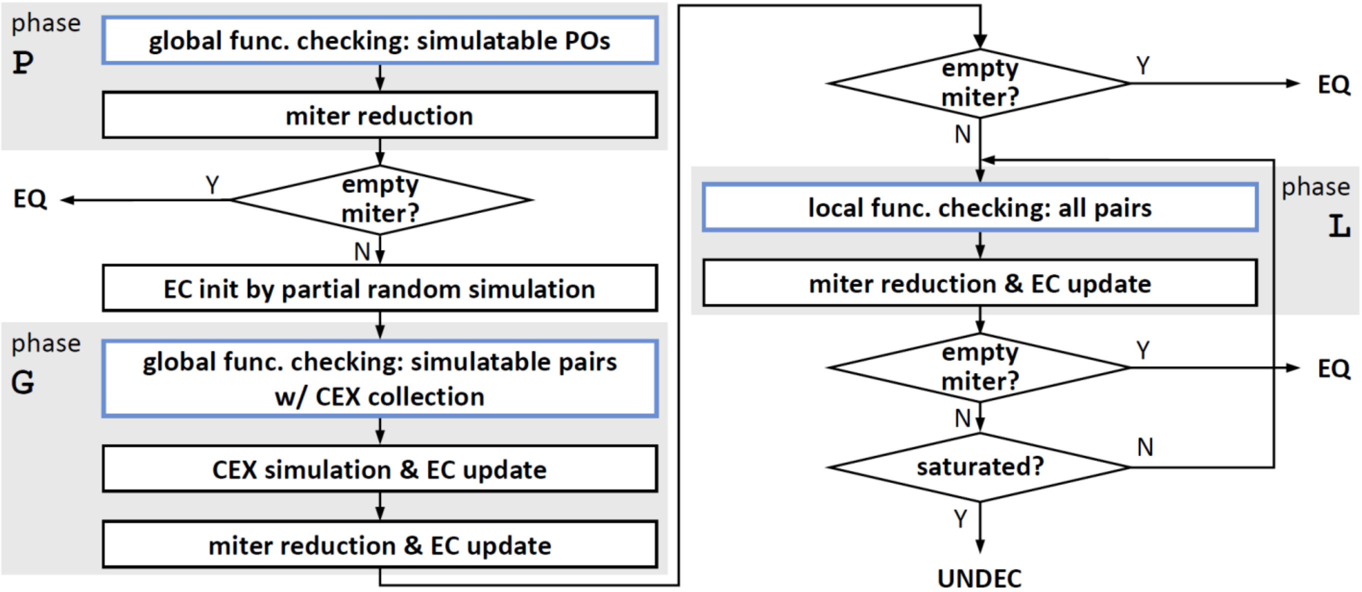
Overall CEC Flow

THE CHIPS TO SYSTEMS CONFERENCE

SPONSORED BY

Overall CEC Flow

- Three types of phases
  - PO checking: check the global functions of **simulatable** (#support <= a constant) PO pairs
  - Global function checking: check the global functions of all simulatable internal node pairs
  - (Until saturation) local function checking: check the local functions of all internal node pairs
- If undecided in the end (i.e., miter not empty), call the CEC engine in ABC



Experimental Results

Setup

- Developed on top of CULS: https://github.com/cuhk-eda/CULS
  - ~8000 lines of code
- Hardware: NVIDIA RTX A6000 GPU, 48 GB DRAM
- Benchmarks: EPFL Combinational Suite and IWLS 2005
  - Miter preparation: optimized circuits generated by ABC `resyn2`
- Use ABC checker for handling undecided case: `&cec -C 100000`
- Compare with
  - Standalone ABC checker
  - Commercial checker: Cadence Conformal LEC (16 CPU threads)

Experimental Results

Benchmarks	Statistics				ABC &cec Runtime (s)	Cfm (16 CPUs) Runtime (s)	Ours (GPU+ABC)				Speed-up	
	#PIs*	#POs*	#Nodes†	Levels†			GPU (s)	Reduced (%)	ABC (s)	Total (s)	vs. ABC	vs. Cfm
hyp_7xd	32768	16384	45881216	24801	7859.26	406002	4616.56	40.2	418.48	5035.04	1.56×	80.64×
log2_10xd	32768	32768	62072832	444	>4 months‡	118392	119633.18	100.0	-	119633.18	88.11×	0.99×
multiplier_10xd	131072	131072	52600832	274	2370.52	3213	159.54	100.0	-	159.54	14.86×	20.14×
sqrt_10xd	131072	65536	44978176	5058	20640.56	30605	52.29	0.7	20623.24	20675.53	1.00×	1.48×
square_10xd	65536	131072	33442816	250	1021.40	2710	144.35	100.0	-	144.35	7.08×	18.77×
voter_10xd	1025024	1024	21862400	70	62610.44	1166	54.20	43.5	35611.63	35665.83	1.76×	0.03×
sin_10xd	24576	25600	10689536	225	2499.28	2081	78.88	100.0	-	78.88	31.68×	26.38×
ac97_ctrl_10xd	2307072	2299904	22685696	12	248.57	1563	97.51	98.9	22.43	119.94	2.07×	13.03×
vga_lcd_5xd	549248	549856	6337536	24	95.82	317	18.51	20.1	81.95	100.46	0.95×	3.16×
Geomean											4.89×	4.88×

- Independently verifies 4 out of 9 cases (i.e., w/o ABC)
  - Up to 88.11x speed-up vs. ABC (4 months -> 1.4 days)
- When combined with ABC, averaged 4.88x speed-up vs. Conformal LEC

Summary

THE CHIPS TO SYSTEMS CONFERENCE

Summary

- A new perspective of checking equivalences using parallel exhaustive simulation
- Local function checking scheme for reducing checking effort
- Efficient parallel algorithms for exhaustive simulation and local function checking

- 4.89x and 4.88x averaged acceleration over standalone ABC checker and Conformal LEC