THE CHIPS TO SYSTEMS CONFERENCE
SHAPING THE NEXT GENERATION OF ELECTRONICS
JUNE 23-27, 2024
MOSCONE WEST CENTER
SAN FRANCISCO, CA, USA

Massively Parallel AIG Resubstitution

Yang Sun*, **Tianji Liu***, Martin D.F. Wong and Evangeline F.Y. Young
CUHK

Outline

- Motivation & Background
- Parallel AIG Resubstitution
- Experimental Results
- Summary

Motivation

- Logic Optimization
  - Netlists are restructured and simplified to improve metrics, e.g., area and delay
- Research trend of parallel logic optimization
  - Multi-core CPU: AIG rewriting [1]
  - GPU: AIG rewriting [2][3], refactoring, balancing [4]

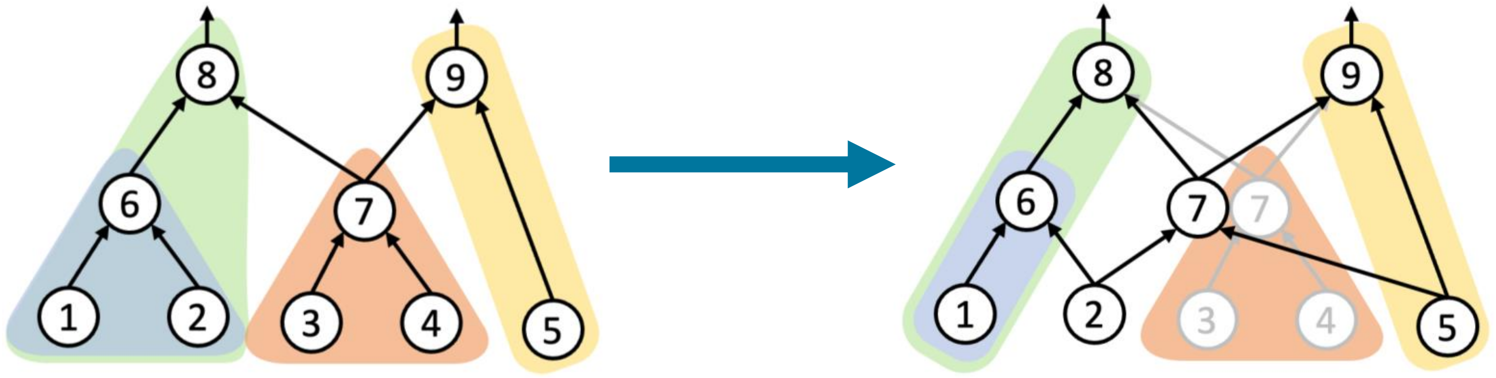
- Resubstitution
  - A more flexible framework than rewriting and refactoring
  - Supports high-effort optimization

[1] V. Possani et al., "Unlocking Fine-grain Parallelism for AIG Rewriting", Proc. ICCAD'18.
[2] S. Lin et al., "NovelRewrite: Node-level Parallel AIG Rewriting", Proc. DAC'22.
[3] L. Li et al., "A Recursion and Lock Free GPU-Based Logic Rewriting Framework Exploiting Both Intranode and Internode Parallelism", IEEE TCAD, 2023.
[4] T. Liu et al., "Rethinking AIG Resynthesis in Parallel", Proc. DAC'23.

Background

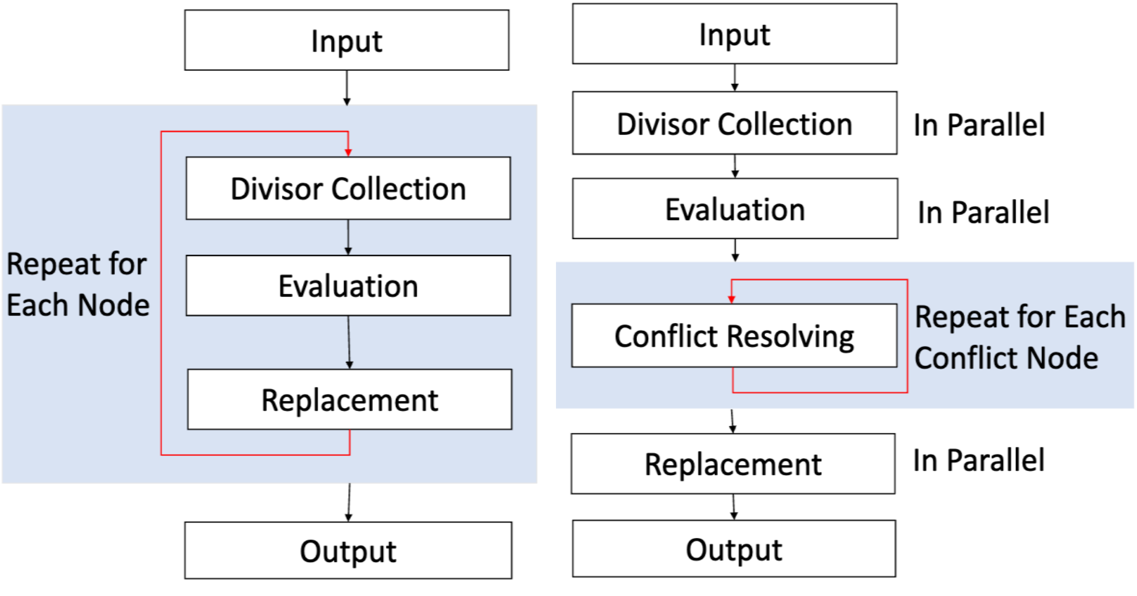
- Resubstitution
  - Re-expresses the function of a node (pivot) using other nodes (divisors) in the logic network
  - The nodes dedicated to driving the pivot (fanout-free cone, FFC) can be removed
- K-resubstitution
  - Adding k new nodes to express the function of the pivot using $k+1$ divisors
  - If FFC size $> k$ (positive gain), the circuit size is reduced
  - Window-based: restrict the candidate divisors in a local region around the pivot

$k = 1$



Overall Flow

- Divisor Collection and Evaluation
  - The most time-consuming procedures
  - Process all nodes in parallel
  - A divisor collection strategy to ensure cycle-freeness
- Conflict Resolving
  - Conflict may occur in parallel replacement
  - Efficient conflict-resolving algorithm
- Replacement
  - Commit the updates in parallel without data race

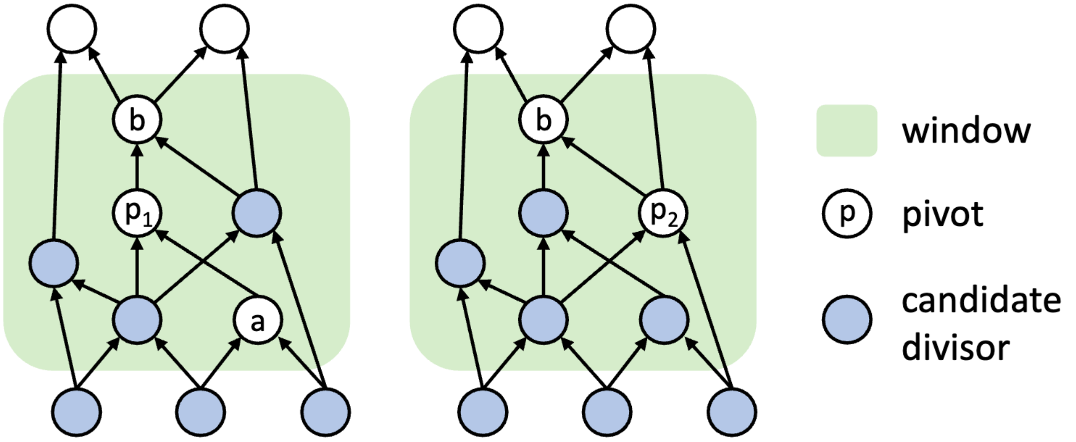


(a) Sequential workflow

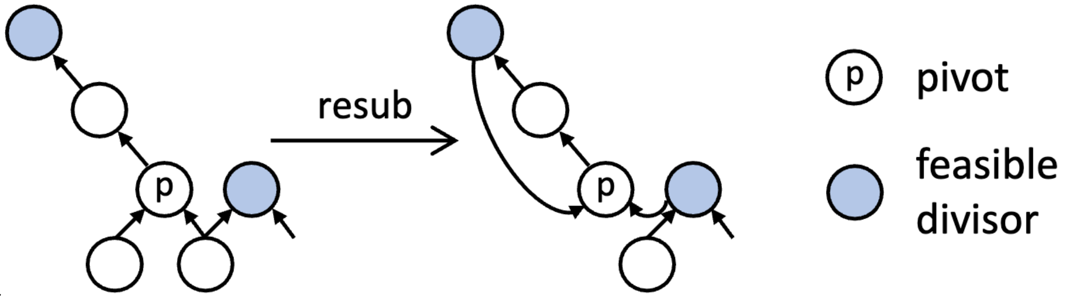
(b) Our workflow

Cycle-Free Divisor Collection

- Window construction
  - Start with a reconvergence-driven cut of pivot
  - Iteratively expand the window toward POs
  - Stop when max window size is reached



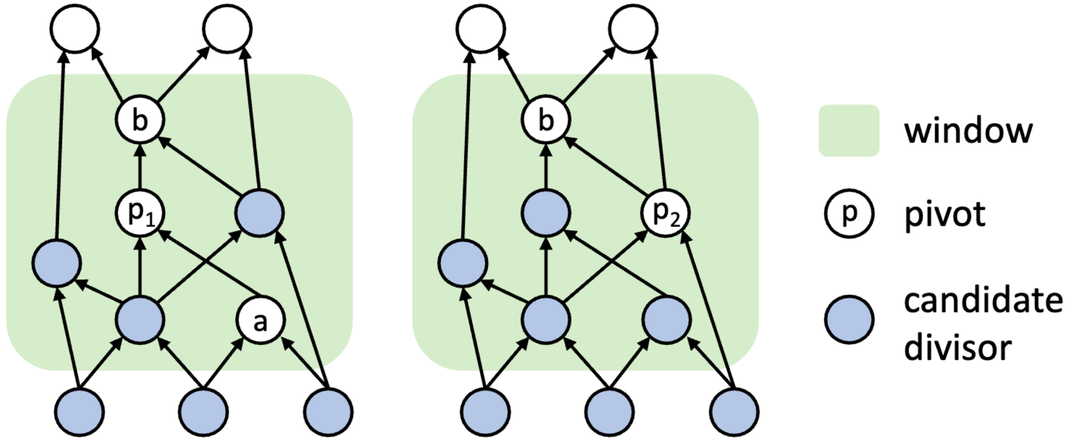
- Divisor Collection
  - In sequential case,
    - Divisors = window – FFC of pivot – TFO of pivot
    - Correctness is guaranteed
  - In parallel case, there may still be cycles
    - E.g., node n is a divisor of m, m is a divisor of n

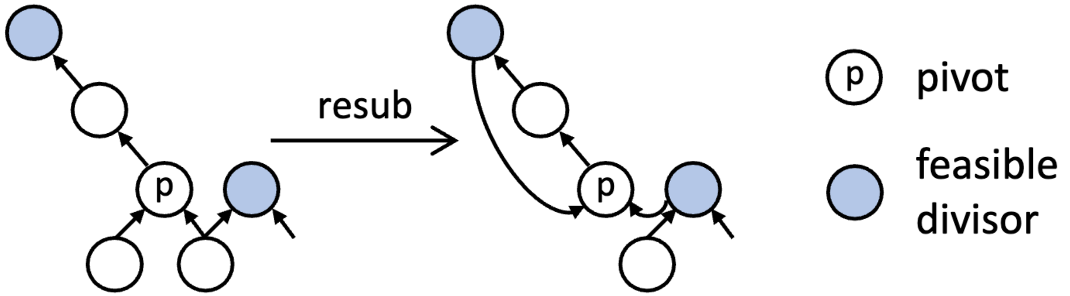


Cycle-Free Divisor Collection

- Resolve cycles before replacement?
  - Time-consuming
  - Potential quality degradation

- Our cycle-free divisor collection strategy
  - Theoretically prevents cycle in parallel case
    - No cyclic dependency if the divisors:
    1. have smaller levels; or
    2. have the same level, but smaller ids





Candidate Divisor Evaluation

- Intuitive Method
  - Exhaustively try all the combinations
- Candidate Filtering
  - Some candidate divisors are trivially infeasible

    Example ($k = 1$): pivot = 0011

    a = 0010   b = 0001   c = 1011   d = 0111   <u>e = 0101</u>

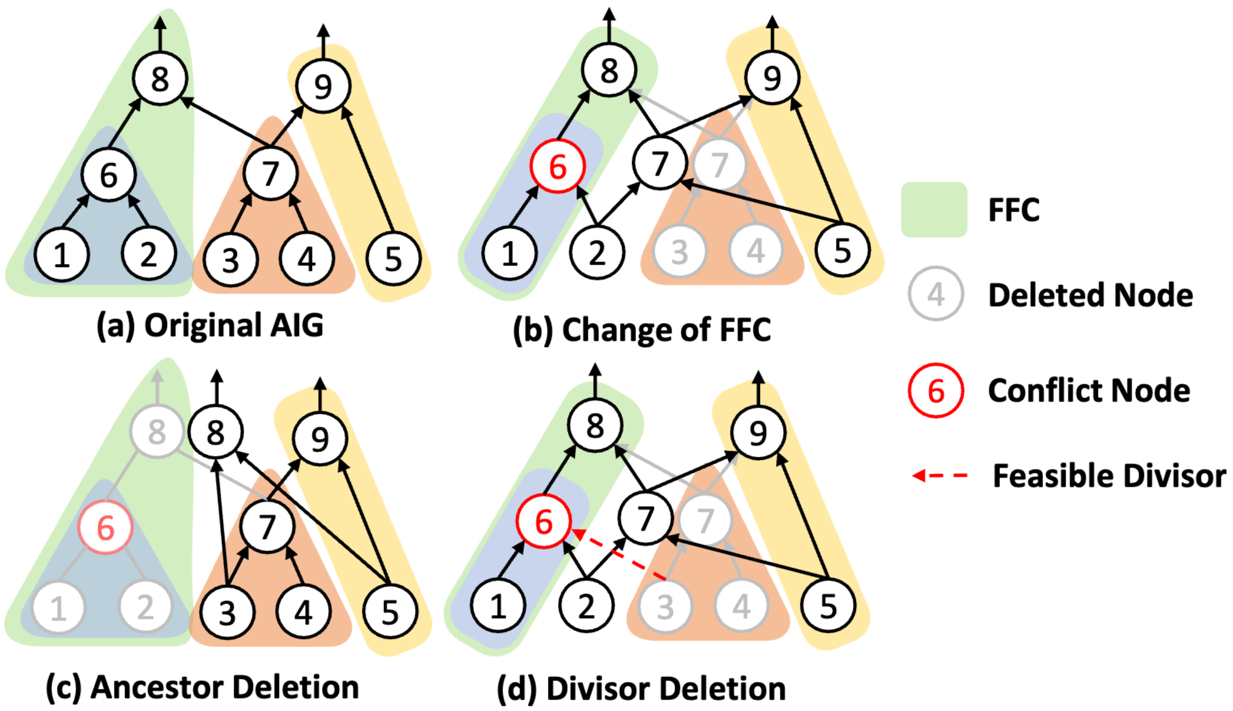
    pivot = a ∨ b = c ∧ d
  - Can be extended to $k > 1$

2-resub case

$$f = \begin{cases} a \vee b \vee c & \text{only if } a \rightarrow f, b \rightarrow f, c \rightarrow f; \\ a \wedge b \wedge c & \text{only if } f \rightarrow a, f \rightarrow b, f \rightarrow c; \\ a \vee (b \wedge c) & \text{only if } a \rightarrow f, (b \wedge c) \rightarrow f; \\ a \wedge (b \vee c) & \text{only if } f \rightarrow a, f \rightarrow (b \vee c). \end{cases}$$

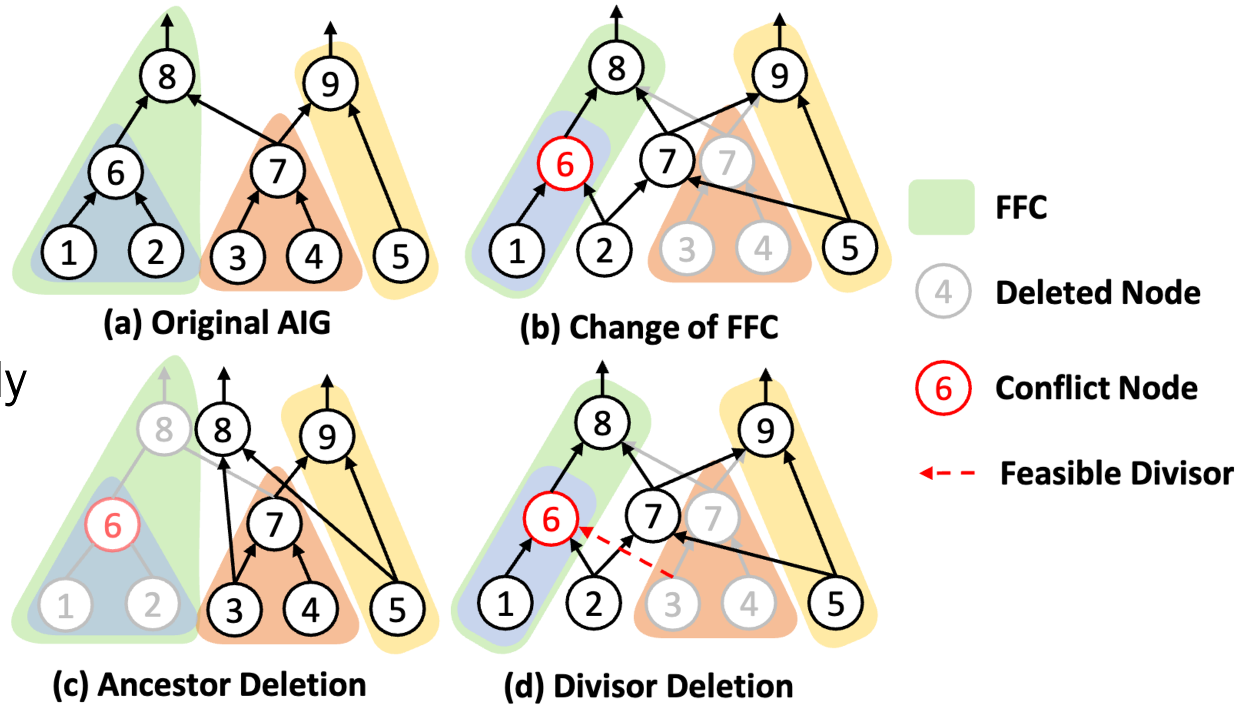
Conflict Resolving

- Issue
  - Still, there are other types of potential conflicts
- Conflict Cases
  - Change of FFC
  - Ancestor Deletion
  - Divisor Deletion
- Two-stage Conflict Resolving
  1. Parallel checking
  2. Sequential Resolving
  - No need to check cycles



Conflict Resolving

- Parallel Checking
  - Conservative assumption that all replacements are accepted
  - Mark FFC nodes as deleted if no conflict
- Sequential Resolving
  - Decide whether to commit an update sequentially
  - If pivot or any divisor deleted (fig. c & d), reject
  - Recompute the pivot's FFC (fig. b)
  - Accept update if gain>0, mark FFC as deleted
- Update the optimized AIG
  - The status of each node is determined
  - Parallel update without data race



Experimental Results – K-Resubstitution

- Benchmarks
  - EPFL Combinational Suite and IWLS 2005 Benchmarks
- Results
  - 41.9x over ABC; 50.3x over mockturtle; best area and second-best delay

Benchmarks	Statistics		ABC resub			mockturtle aig_resub			GPU resub		
	#Nodes	Levels	#Nodes	Levels	Time (s)	#Nodes	Levels	Time (s)	#Nodes	Levels	Time (s)
sixteen	16216836	140	16143196	140	235.9	16173172	140	275.3	16140545	140	5.0
twenty	20732893	162	20648660	162	317.8	20680795	162	370.2	20648313	162	6.3
twentythree	23339737	176	23249547	176	357.8	23283697	176	431.7	23249102	176	7.6
div_10xd	58620928	4372	46350336	4372	798.0	46245888	4402	1537.8	46311424	4405	22.6
hyp_8xd	54869760	24801	53232640	24801	778.2	53149696	24804	448.6	54393600	24802	25.3
mem_ctrl_10xd	47960064	114	47481856	114	631.1	47731712	114	748.9	47640576	114	10.5
log2_10xd	32829440	444	32001024	433	554.9	32043008	435	456.9	31578112	423	11.4
multiplier_10xd	27711488	274	26878976	273	394.1	26966016	273	441.1	26624000	272	8.6
sqrt_10xd	25208832	5058	21885952	5058	325.3	21013504	5900	438.6	21151744	5990	11.0
square_10xd	18927616	250	17263616	250	264.9	18396160	250	238.4	17537024	250	5.9
voter_10xd	14088192	70	9511936	65	161.5	10733568	73	136.0	9764864	69	3.9
sin_10xd	5545984	225	5372928	225	82.3	5428224	227	87.4	5354496	223	2.1
ac97_ctrl_10xd	14610432	12	14294016	12	176.1	13800448	12	862.0	13766656	12	4.4
vga_lcd_5xd	4054752	24	3809056	24	133.8	3809984	24	165.8	3810368	24	3.6
Geomean Ratio			1.000	1.000	41.9	1.009	1.021	50.3	0.998	1.014	1.0

Experimental Results – Optimization Sequence

- Fully GPU-parallelized sequence `resyn2rs`
  - GPU-based balancing, rewriting and refactoring in CULS [5]
  - Integrate them with our GPU resub
- 0.8% smaller area, 5.8% smaller delay, 46.4x acceleration

Benchmarks	ABC `resyn2rs`			GPU `resyn2rs`*		
	#Nodes	Levels	Time (s)	#Nodes	Levels	Time (s)
sixteen	11970378	99	8530.2	11781381	64	63.2
twenty	15309087	86	9763.8	15106639	65	81.2
twentythree	17160203	94	11809.5	16942499	68	91.8
div_10xd	41717760	4370	12581.6	41646619	4413	381.2
hyp_8xd	52361984	24792	25026.0	53181303	24671	775.2
mem_ctrl_10xd	44986368	112	13269.5	43365698	91	403.1
log2_10xd	29893632	376	12815.1	29998592	357	339.3
multiplier_10xd	24922112	262	8505.2	24968192	262	184.2
sqrt_10xd	19584000	4968	6979.8	18688000	5927	206.6
square_10xd	16272384	248	5409.1	16237303	246	108.0
voter_10xd	8138752	57	3026.8	8220679	61	48.7
sin_10xd	5141504	175	2116.4	5143988	165	78.4
ac97_ctrl_10xd	10604544	9	2321.0	10526720	9	65.8
vga_lcd_5xd	2906272	18	2302.2	2903809	24	135.5
Geomean Ratio	1.000	1.000	46.4	0.992	0.942	1.0

[5] https://github.com/cuhk-eda/CULS

Experimental Results – Divisor Collection Strategies

- Comparing three strategies
  - Our cycle-free divisor collection strategy
  - Full divisor collection with additional cycle checking
  - Only collect divisors from smaller levels
- Our strategy is efficient with preferable quality

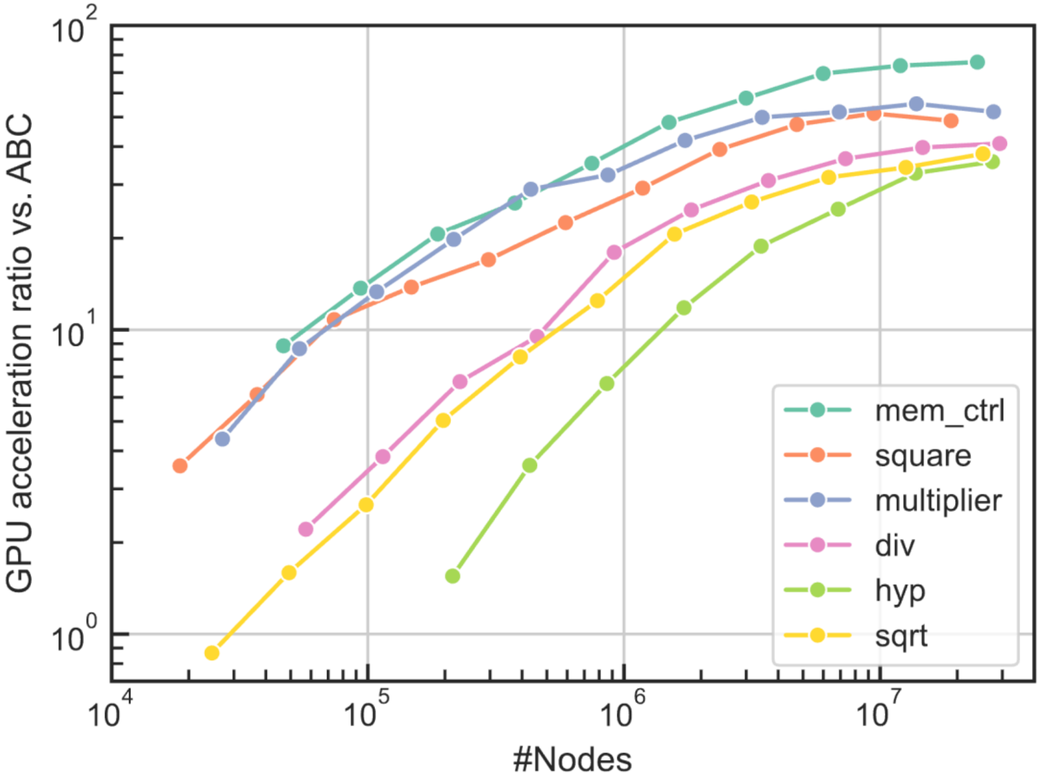
Our cycle-free strategy:

1. divisors with smaller levels; or,
2. with same level, but smaller ids

Method	Norm. #Nodes	Norm. Time
ours	1.000	1.0
full+resolve_cycle	0.997	2.3
smaller_level	1.010	1.0

Experimental Results – Scalability

- Speedup with different AIG sizes



Summary

- Propose an efficient GPU-parallel framework for window-based k-resubsitution
- 41.9× and 50.3× acceleration over ABC and mockturtle on large AIG benchmarks, with comparable or better qualities
- 46.4× with superior optimization quality over ABC on the `resyn2rs` sequence

- Open-sourced in CULS: https://github.com/cuhk-eda/CULS