Rethinking AIG Resynthesis in Parallel

Tianji Liu and Evangeline F.Y. Young

The Chinese University of Hong Kong

Outline

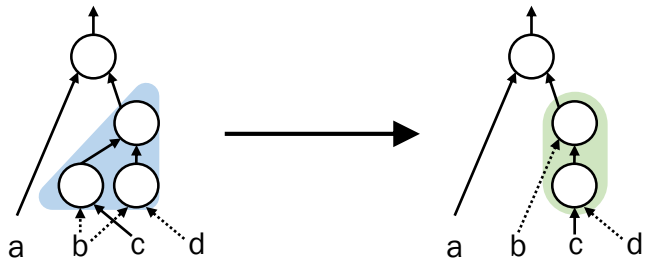
- Motivation
- AIG Refactoring
  - Background
  - Proposed Method - Parallel Refactoring
- AIG Balancing
  - Background
  - Proposed Method - Parallel Balancing
- Experimental Results
- Summary

Motivation

- AIG resynthesis is equipped with many algorithms
  - rewrite (`rw`), refactor (`rf`), (AND-)balance (`b`), etc.
  - Different algorithm applies different optimization strategy
- A commonly used AIG resynthesis flow
  - `(resyn2) b; rw; rf; b; rw; rw -z; b; rf -z; rw -z; b`
- GPU has shown its effectiveness in accelerating rewriting
  - S. Lin et al., "NovelRewrite: node-level parallel AIG rewriting," DAC 2022.
- To fully accelerate `resyn2`, parallel refactor and balance are indispensable

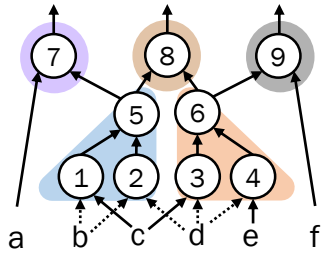
Background - AIG Refactoring

- Optimizes the area of AIG
- Resynthesize large logic cones
  - By algebraic factoring from collapsed SOPs
  - Using one cut per node to reduce time complexity
- Replace the original cone if the resynthesized cone has positive gain

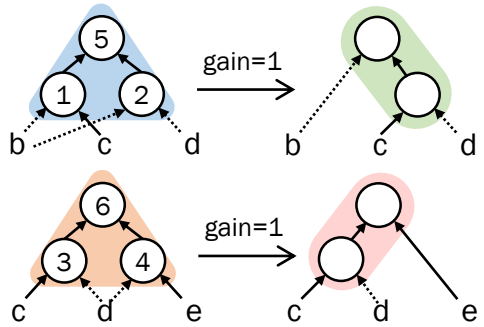


Parallel Refactoring

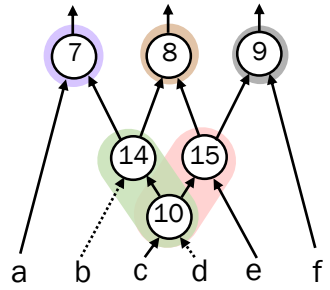
- Big picture
  - Stage 1: identify and collapse logic cones (subgraphs) that are
    - (1) disjoint from each other, and
    - (2) cover all the logic (non-PI nodes) in the AIG
  - Stage 2: resynthesize all local functions in parallel
  - Stage 3: replace the original cones by the resynthesized cones in parallel



identification and collapsing



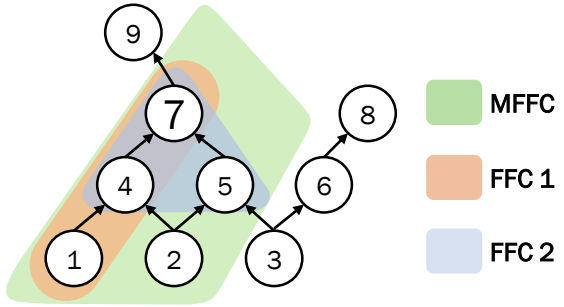
resynthesis



replacement

Parallel Refactoring - Stage 1

- Maximum fanout-free cone (MFFC) of a node
  - Contains all the logic dedicated to driving the node
- Fanout-free cone (FFC) of a node
  - A logic cone that is a subset of the MFFC of the node



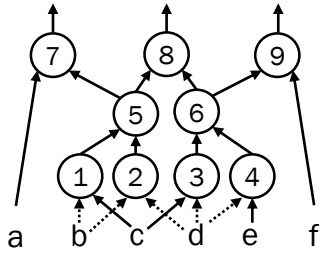
(M)FFCs of node 7

- Property
  - MFFCs of different nodes are either subsets of each other, or disjoint from each other

Parallel Refactoring - Stage 1

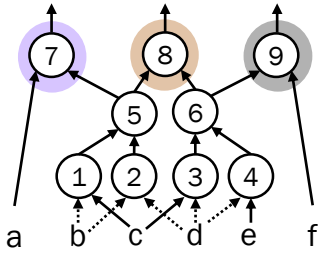
- How to achieve the two targets of stage 1?
- **Level-wise parallel** identification and collapsing
  - Maintain a frontier array, initialized by all POs
  - While there exists non-PI node in the frontier (i.e., for each level)
    - Identify MFFCs rooted at each node in the frontier, one thread per node
    - Gather all the inputs to the MFFCs to be the new frontier

frontier: 7 8 9



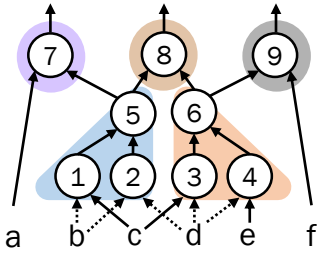
before level 0

frontier: a 5 6 f



before level 1

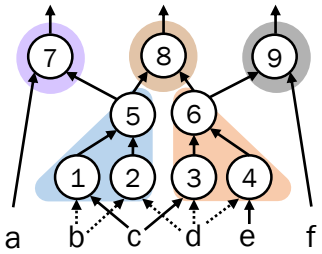
frontier: b c d e



before level 2

Parallel Refactoring - Stage 1

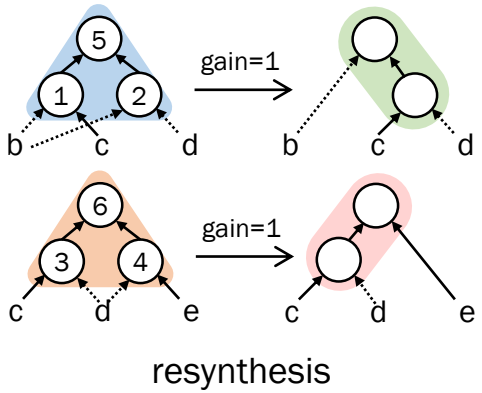
- A practical issue
  - In refactoring, #inputs of a cone is bounded by k to restrict the time complexity
  - What if an MFFC has more than k inputs?
- Our approach
  - Stop identification of an MFFC when reaching k inputs, and thus an FFC is obtained
- Theoretical result
  - It can be proved that, by doing so, the identified FFCs by our level-wise parallel procedure are **disjoint**



identification and collapsing

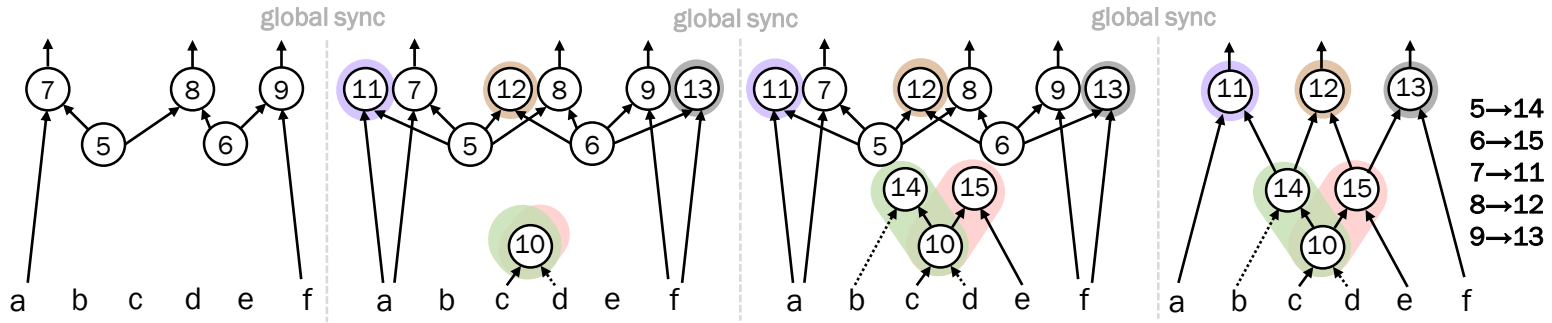
Parallel Refactoring - Stage 2

- Resynthesize all local functions in parallel
  - One thread per function



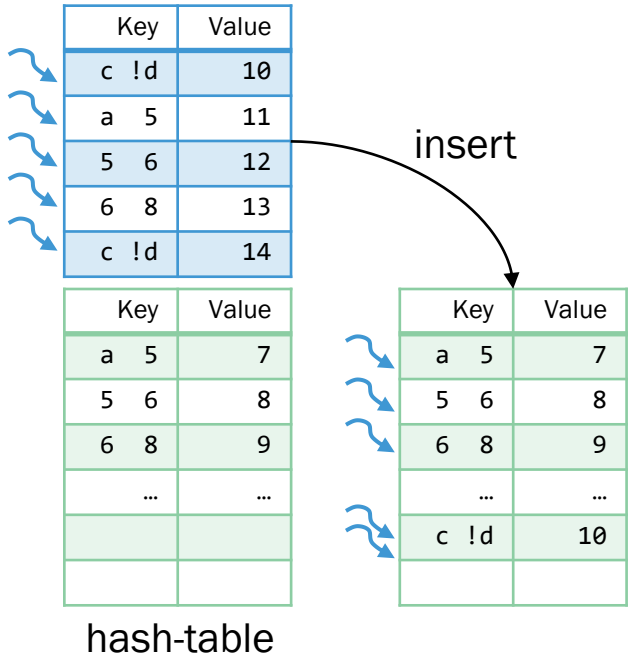
Parallel Refactoring - Stage 3

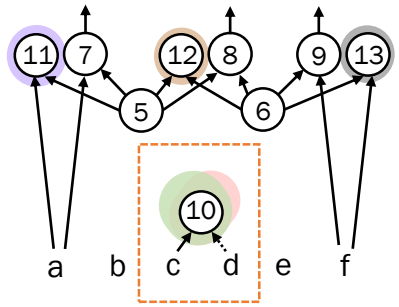
- Update the resynthesized cones in parallel
  - Start with an AIG containing
    - old cones whose corresponding new cone has negative gain, and
    - the cut nodes of the collapsed FFCs
  - Insert the new nodes synchronously in parallel, one thread per cone
  - Replace the old cone roots by the new roots



GPU Hash-table

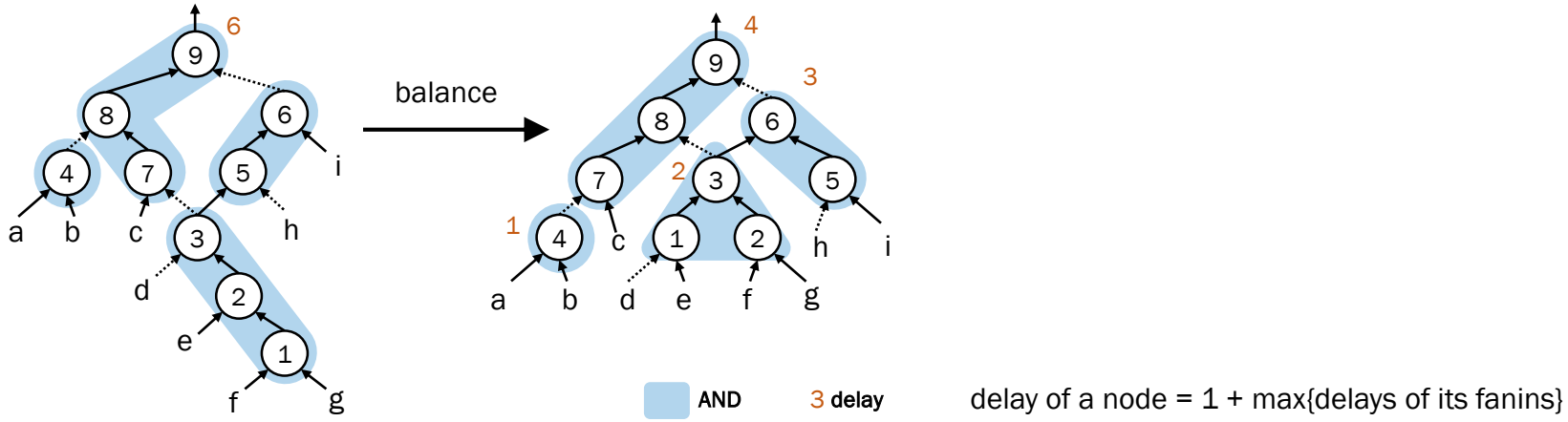
- Enable logic sharing by structural hashing
  - Developed a GPU hash-table supporting batched node insertion and retrieval
  - Ensures that only one node can exist with a particular fanin and negation status





Background - AIG Balancing

- Identify and collapse n-input AND gates with a tree structure
- Recursively balance the cut nodes, with their delays obtained
- Reconstruct the cone by AND-ing the inputs with the order of ascending delay



Parallel Balancing

- Stage 1: identification and collapsing of n-input ANDs
  - Similar to the stage 1 of parallel refactoring, in level-wise parallel from POs to PIs
- Stage 2: reconstruction of n-input ANDs
  - Still in level-wise parallel (from PIs to POs)
  - This is because reconstruction needs the delay of input nodes to be determined first

Experimental Results - Acceleration & QoR

- Tested on enlarged benchmarks from the EPFL and IWLS 2005 Suite
- Similar or slightly better QoR (AIG area and delay) compared with ABC

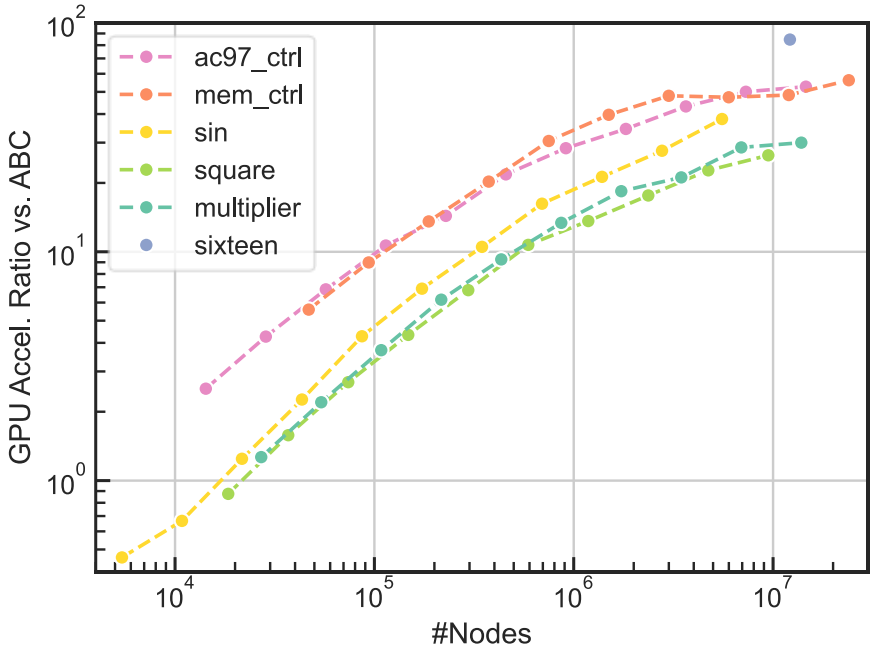
Benchmarks	ABC `rf_resyn`		GPU `rf_resyn`		ABC `resyn2`		GPU `resyn2` (`rwz` ×2)		GPU `rf` (×2)	
	#Nodes / Levels	Time	#Nodes / Levels	Time	#Nodes / Levels	Time	#Nodes / Levels	Time	#Nodes / Levels	Time
twentythree	17396577 / 104	1490.4	17348255 / 98	17.3	16931332 / 72	4174.5	16915942 / 68	55.2	18397973 / 103	10.7
twenty	15514368 / 94	1359.7	15480873 / 90	15.2	15095643 / 65	3633.5	15079948 / 65	49.1	16421342 / 95	9.2
sixteen	12147445 / 99	1115.4	12118520 / 99	13.2	11765351 / 68	2760.7	11757432 / 64	40.4	12911473 / 101	7.6
div_10xd	56788992 / 4404	2273.1	48652485 / 4373	104.7	41665780 / 4388	8028.8	41689305 / 4422	239.8	48779264 / 4422	20.5
hyp_8xd	54539008 / 24785	2104.8	54539008 / 24787	345.2	54193719 / 24785	10771.1	54205696 / 24671	653.8	54539008 / 24790	32.7
mem_ctrl_10xd	46615552 / 105	3448.9	47312818 / 108	54.7	43777052 / 92	6936.8	44821695 / 94	132.9	47444992 / 109	16.9
log2_10xd	31011840 / 366	1434.7	31371816 / 390	34.3	29946093 / 358	4879.4	29966717 / 358	91.5	31552512 / 396	7.9
multiplier_10xd	26471424 / 265	947.1	26469376 / 265	33.0	24957961 / 262	3509.3	24949760 / 262	77.5	26565632 / 265	5.5
sqrt_10xd	23618560 / 5182	3904.8	23014400 / 5174	54.8	18884491 / 6020	5639.8	18800640 / 5928	131.5	24862720 / 5365	12.4
square_10xd	17935360 / 250	606.7	17888256 / 250	22.9	17091593 / 248	2343.3	17052614 / 249	58.0	18081792 / 250	3.7
voter_10xd	9951232 / 59	370.6	10874377 / 63	14.6	8845336 / 66	1432.4	8961831 / 60	33.2	11480064 / 66	2.5
sin_10xd	5285888 / 179	232.0	5319346 / 187	6.2	5156077 / 163	851.0	5158131 / 161	16.2	5357568 / 213	1.5
ac97_ctrl_10xd	10956800 / 11	699.6	11020147 / 9	13.5	10490213 / 10	1375.5	10660403 / 10	32.0	11144192 / 12	3.4
vga_lcd_5xd	2918528 / 18	189.5	2946898 / 20	5.2	2903540 / 24	439.6	2903968 / 23	10.7	2952480 / 26	1.2
Geomean Ratio vs. ABC	1.000 / 1.000	1.0	0.996 / 1.000	39.5× accel.	1.000 / 1.000	1.0	1.003 / 0.982	45.9× accel.	0.983 / 0.980	42.7× accel.

`rf -z;`
`z; b`

te in

Experimental Results - Scalability

- Speedup with different AIG sizes on the `rf_resyn` script
- GPU implementation is faster than ABC when #node > 30k



Experimental Results - Runtime Breakdown

- Balancing is the bottleneck, due to its fully level-wise parallel nature

Summary

- Propose novel parallel algorithms for AIG balancing and refactoring
- 14.8x and 42.7x acceleration over ABC for GPU balancing and refactoring with better QoR
- 45.9x acceleration over ABC for the commonly-used optimization script `resyn2` with comparable QoR