Parallel Gröbner Basis Rewriting and Memory Optimization for Efficient Multiplier Verification

Hongduo Liu[1], Peiyu Liao[1], Junhua Huang[2], Hui-Ling Zhen[2], Mingxuan Yuan[2], Tsung-Yi Ho[1], Bei Yu[1]
[1]The Chinese University of Hong Kong [2]Huawei Noah's Ark Lab

Multiplier Verification

- **Wide applications** of integer multipliers: signal processing, cryptography, scientific computing, etc.
- **Formal** verification is essential to ensure reliability.
- Verifying highly parallelized and structurally complex multipliers is **time-consuming**.

Formal Verification Methods

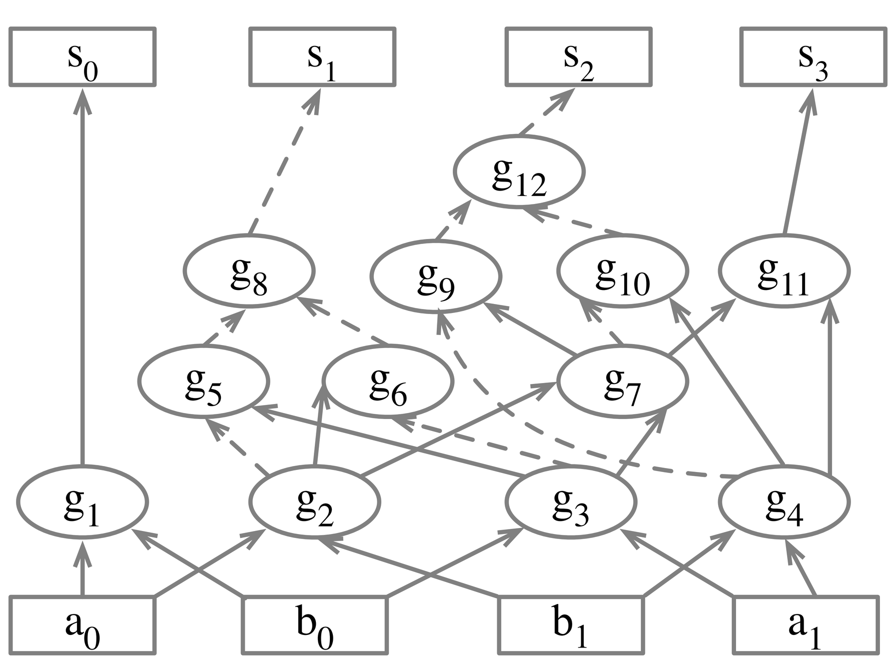
- Binary Decision Tree: memory explosion and structural information dependent.
- SAT: poor scalability when facing large multipliers.
- Symbolic Computer Algebra: achieves SOTA performance.

SCA-based Verification

- Step 1: Gröbner basis construction.
- Step 2: Gröbner basis rewriting.
- Step 3: Specification polynomial reduction.
- Step 4: Zero remainder implies correctness.

Gröbner Basis Construction

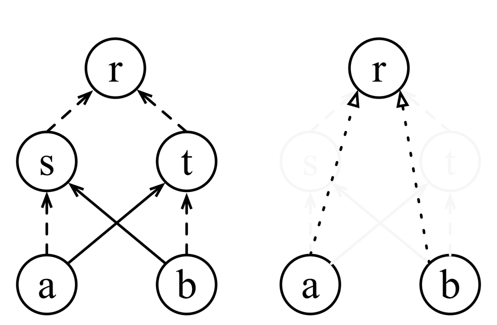
Model the circuit as Gröbner basis polynomials $G = \{f_{g_1}, \ldots, f_{g_n}\}$ and the specification as a polynomial sp.



$$sp := 8s_3 + 4s_2 + 2s_1 + s_0 - (2a_1 + a_0)(2b_1 + b_0)$$
$$f_{s_3} := -s_3 + g_{11}$$
$$f_{g_{11}} := -g_{11} + g_4 g_7$$
$$f_{s_2} := -s_2 + 1 - g_{12}$$
$$f_{g_{12}} := -g_{12} + 1 - g_9 - g_{10} + g_9 g_{10}$$
$$\ldots$$

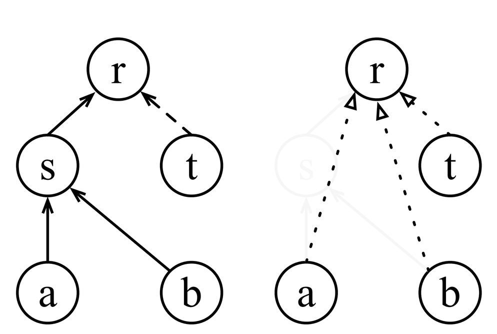
Gröbner Basis Rewriting

The objective of Gröbner basis rewriting is to acquire a new basis with fewer variables, preventing the blow-up of monomials.



- Before rewriting: $f_r := -r + 1 - r - s + st$, which depends on s and t.
- After rewriting: $f_r := -r + a + b - 2ab$, which depends on a and b.

XOR-Rewriting [5] removes all variables that are neither an input nor an output of an XOR-gate.



- Before rewriting: $f_r := -r + s - st$, which depends on s and t.
- After rewriting: $f_r := -r + ab - abt$, which depends on a and b.

Common-Rewriting [5] simplifies the Gröbner basis by eliminating all gates that only possess a single fanout.

Specification Polynomial Reduction

Suppose we desire to reduce $sp = c_1x_1x_2x_3 + \cdots + c_2x_2x_4$ with a polynomial $f_{x_2} = -x_2 + c_3x_5x_6$ from simplified Gröbner basis. The reduction process can be accomplished by the following steps:

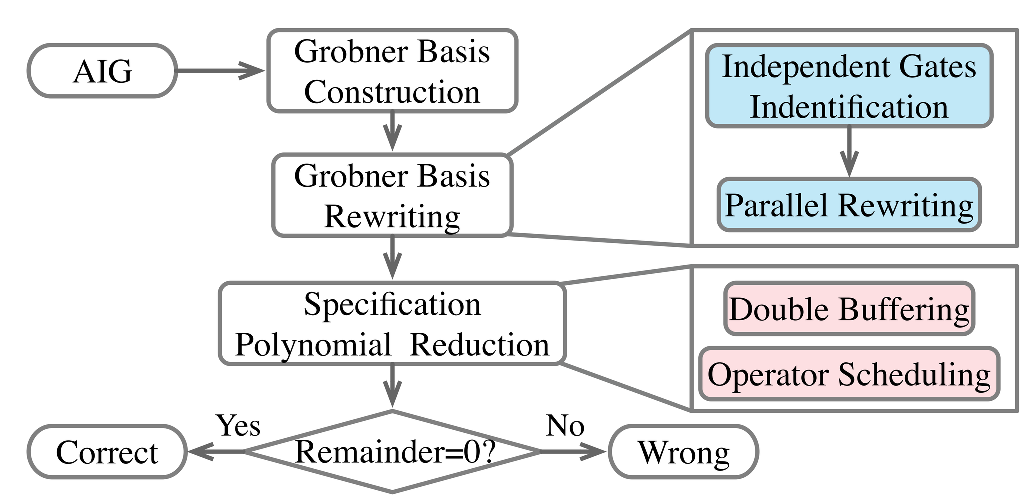
- **Divide term**: identify all terms containing the variable x_2 in sp, divide x_2 from those terms and add them together to get $quo := c_1x_1x_3 + c_2x_4$.
- **Multiply poly**: multiply the obtained quo with f_{x_2}, resulting in a polynomial $mul := -c_1x_1x_2x_3 + c_1c_3x_1x_3x_5x_6 - c_2x_2x_4 + c_2c_3x_4x_5x_6$.
- **Add poly**: add mul to sp to cancel all terms containing the variable x_2, which are $c_1x_1x_2x_3$ and $c_2x_2x_4$.

Previous Works

- Reduce the verification complexity by detecting redundant polynomials [6].
- Allow for local cancellation of vanishing monomials in converging gates cones starting from half adders [3].
- Substitute complex final-stage adders with simple ripple-carry adders and used SAT solvers to verify the equivalence of substitution [2].

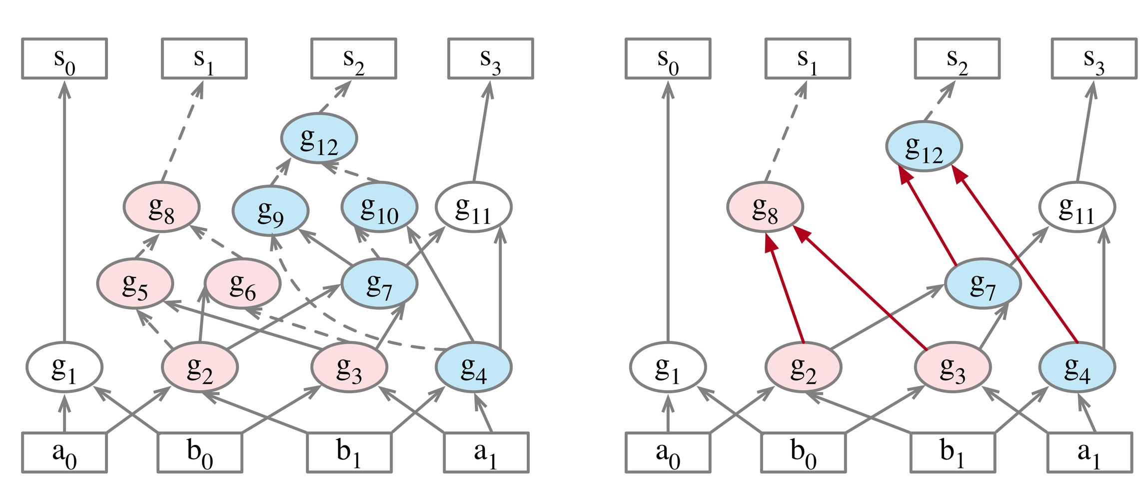
Whole Flow of Our Framework

Key contribution: accelerating verification by parallel computing and memory footprint optimization.



Parallel Gröbner Basis Rewriting

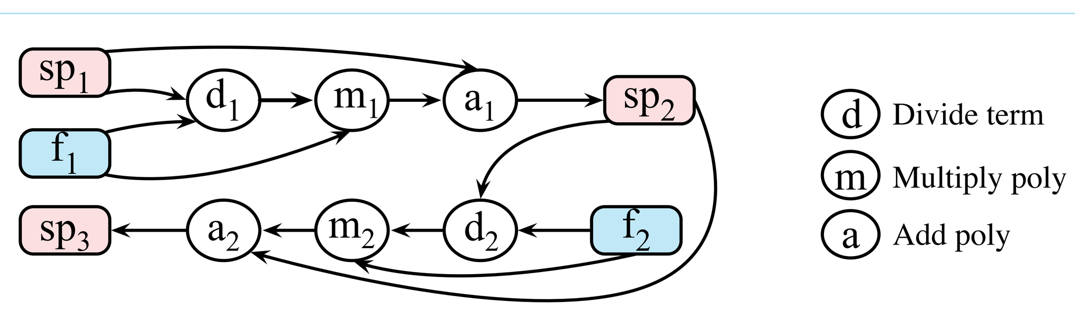
- We observe that the elimination of certain variables can operate independently of others.
- The elimination of (g_5, g_6) and (g_9, g_{10}) in the AIG are independent of each other and thus can be done in parallel.



$$f_{g_8} := -g_8 + 1 - g_5 - g_6 + g_5g_6$$
$$f_{g_5} := -g_5 + g_3 - g_2g_3$$
$$f_{g_6} := -g_6 + g_2 - g_2g_3$$
$$\ldots$$

$$f_{g_8} := -g_8 - g_2 - g_3 + 2g_2g_3$$
$$f_{g_{12}} := -g_{12} - g_4 - g_7 + 2g_4g_7$$
$$\ldots$$

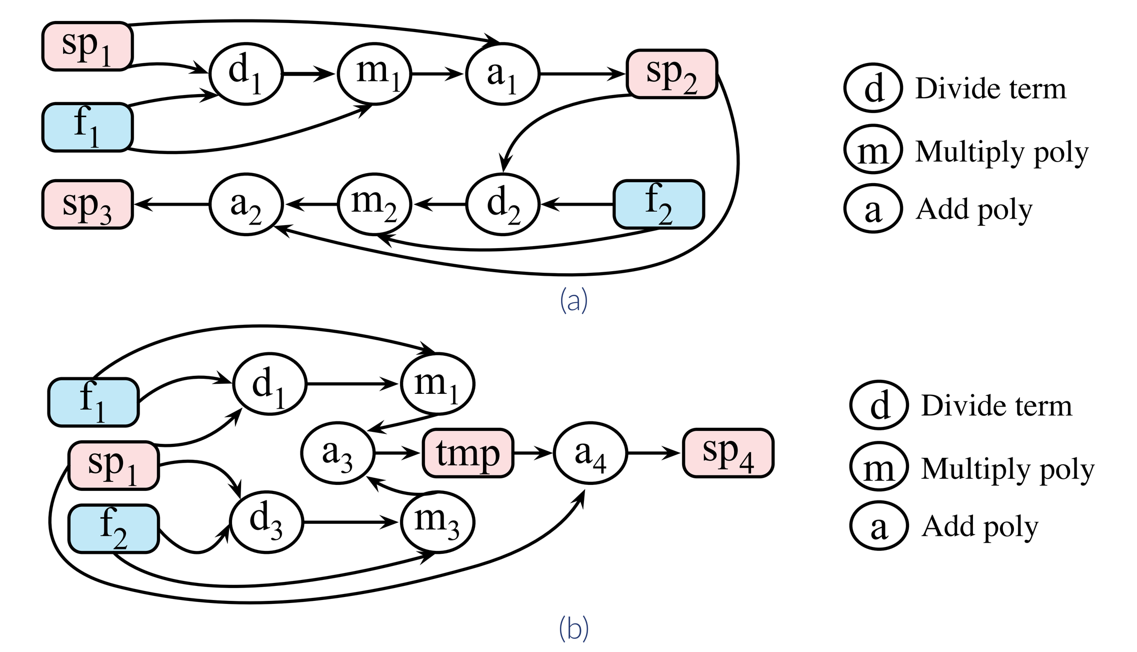
Double Buffering



- Cache the first polynomial sp_1 in the first buffer.
- After reducing it by f_1, the derived polynomial, sp_2, is stored in the second buffer.
- sp_1 is no longer needed, so the first buffer can be used to store the newly derived polynomial, sp_3.

Operator Rescheduling

Suppose we have $f_1 := -\alpha + h(T_\alpha)$ and $f_2 := -\beta + h(T_\beta)$. $h(T_\alpha)$ and $h(T_\beta)$ are both polynomials. If $\alpha \notin h(T_\beta)$, $\beta \notin h(T_\alpha)$ and $\forall u \in sp_1, u \xrightarrow{\alpha\beta} r \neq 0$, where u is a monomial in sp_1, then the reduction of f_1 and f_2 can be performed concurrently.



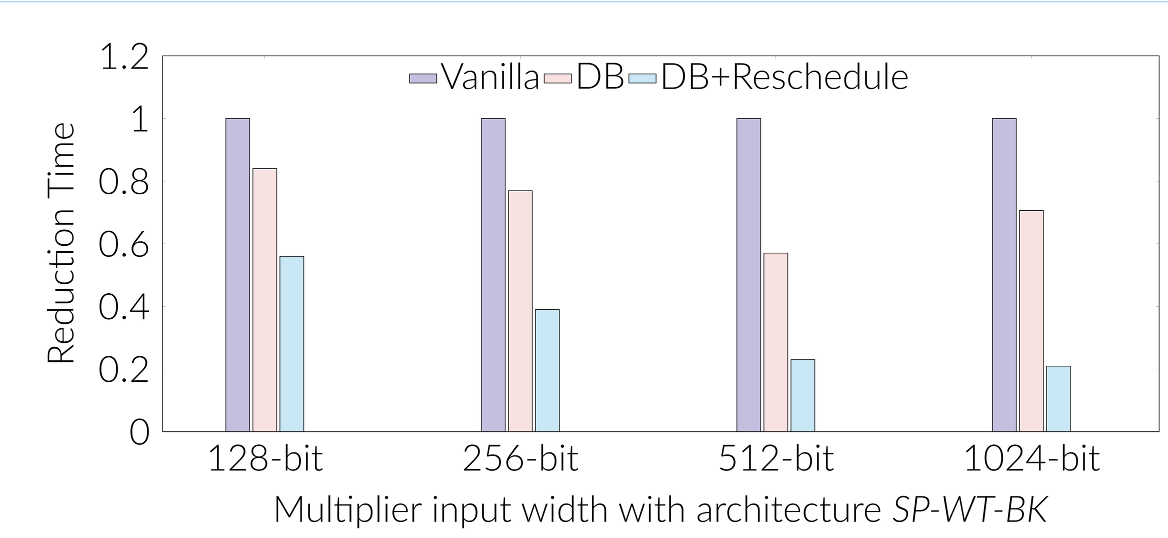
The second computation graph exhibits smaller memory overhead because tmp is a shorter polynomial than $sp2$.

Experimental Results

Verification runtime comparison on large multipliers generated by GenMul [4].

benchmark	size	#gates	Amulet 2.2 [1] rewriting	Amulet 2.2 [1] reduction	Amulet 2.2 [1] overall	Ours (16 threads) rewriting	Ours (16 threads) reduction	Ours (16 threads) overall
SP-AR-LF	128×128	194314	0.98	0.16	1.30	0.43	0.57	1.15
SP-DT-LF		193806	2.26	0.38	2.82	0.45	0.82	1.39
SP-WT-BK		197774	2.31	2.65	5.24	0.48	1.26	1.92
SP-AR-LF	256×512	781834	6.20	0.87	7.72	2.37	1.52	4.55
SP-DT-LF		780814	17.85	2.09	20.80	1.79	3.57	6.06
SP-WT-BK		790610	17.84	25.85	44.66	1.86	5.63	8.16
SP-AR-LF	512×512	3136522	55.42	5.70	63.81	10.94	6.97	20.13
SP-DT-LF		3134478	185.53	12.02	201.13	8.28	15.22	26.33
SP-WT-BK		3157890	186.46	322.87	512.96	8.84	33.05	45.02
SP-AR-LF	1024×1024	12564490	506.55	39.39	573.05	54.26	37.41	102.11
SP-DT-LF		12560398	1817.96	92.74	1940.23	38.32	73.96	123.68
SP-WT-BK		12606714	1807.25	3519.13	5356.14	37.65	311.19	360.71
Average Ratio			15.64	2.80	6.24	1.00	1.00	1.00

Effectiveness of Memory Optimization Techniques



Both DB (double buffering) and operator rescheduling are useful for reducing memory overhead.

References

[1] Daniela Kaufmann and Armin Biere. Fuzzing and delta debugging and-inverter graph verification tools. In *International Conference on Tests and Proofs*, pages 69–88. Springer, 2022.

[2] Daniela Kaufmann, Armin Biere, and Manuel Kauers. Verifying large multipliers by combining SAT and computer algebra. In *Proc. FMCAD*, pages 28–36. IEEE, 2019.

[3] Alireza Mahzoon, Daniel Große, and Rolf Drechsler. Polycleaner: clean your polynomials before backward rewriting to verify million-gate multipliers. In *Proc. ICCAD*, pages 1–8. IEEE, 2018.

[4] Alireza Mahzoon, Daniel Große, and Rolf Drechsler. Genmul: Generating architecturally complex multipliers to challenge formal verification tools. In *Recent Findings in Boolean Techniques*, pages 177–191. Springer, 2021.

[5] Amr Sayed-Ahmed, Daniel Große, Ulrich Kühne, Mathias Soeken, and Rolf Drechsler. Formal verification of integer multipliers by combining gröbner basis with logic reduction. pages 1048–1053. IEEE, 2016.

[6] Cunxi Yu, Maciej Ciesielski, and Alan Mishchenko. Fast algebraic rewriting based on and-inverter graphs. *IEEE TCAD*, 37(9):1907–1911, 2017.

hdliu21@cse.cuhk.edu.hk