Gamora: Graph Learning based Symbolic Reasoning for Large-Scale Boolean Networks

Nan Wu¹, Yingjie Li², Cong Hao³, Steve Dai⁴, Cunxi Yu², Yuan Xie⁵

¹University of California, Santa Barbara, ²University of Utah, ³Georgia Institute of Technology, ⁴NVIDIA, ⁵Alibaba DAMO Academy
Outline

- Motivation
- Problem formulation
  - What is Gamora?
  - Why Gamora?
  - How Gamora?
- Evaluation
  - High reasoning accuracy
  - Scalability
  - Generalization
Verification is Important!

Hardware and Software Development Efforts for Advanced Designs

Cost ($M)

Verification takes half of hardware development cost!

Technologies node

Verification is Important!

Percentage of Project Time Spent in Verification

Veriﬁcation dominates the project time!

Functional Unit Identification

Identifying functional units from flattened gate-level netlists has wide applications
Functional Unit Identification

Identifying functional units from flattened gate-level netlists has wide applications

- Word-level abstraction for formal verification

```java
int OUT = [
  s : A;
  1 : B;
];
```
Functional Unit Identification

Identifying functional units from flattened gate-level netlists has wide applications

- Word-level abstraction for formal verification
- Functional verification
  - [TCAD’17] Fast algebraic rewriting based on and-inverter graphs
  - [ICCAD’18] PolyCleaner: clean your polynomials before backward rewriting to verify million-gate multipliers
  - [DAC’19] RevSCA: Using reverse engineering to bring light into backward rewriting for big and dirty multipliers
  - [TCAD’19] Understanding algebraic rewriting for arithmetic circuit verification: a bit-flow model
Functional Unit Identification

Identifying functional units from flattened gate-level netlists has wide applications
- Word-level abstraction for formal verification
- Functional verification
- Logic optimization and datapath synthesis
  - [DATE’15] A universal macro block mapping scheme for arithmetic circuits
Functional Unit Identification

Identifying functional units from flattened gate-level netlists has wide applications

- Word-level abstraction for formal verification
- Functional verification
- Logic optimization and datapath synthesis
- Hardware trojan detection
  - [DAC’19] Attacking split manufacturing from a deep learning perspective
  - [JETC’21] Hardware trust and assurance through reverse engineering: A tutorial and outlook from image analysis and machine learning perspectives
Challenges in Conventional Methods

- **Structural methods**
  - Focus on circuit topology
  - Efficient with customized algorithms ✓
  - Often mathematically incomplete, rely heavily on reference circuits ✗
  - Memory-consuming for large Boolean networks with billions of nodes ✗

- **Functional methods**
  - Functionally analyze the circuit for potential arithmetic components
  - Accurate and solver-ready ✓
  - Ultra-long runtime ✗
  - Example: [TETC’13] Pramod Subramanyan et al. Reverse engineering digital circuits using structural and functional analyses
Solution: Gamora

Graph Learning based Symbolic Reasoning for Large-Scale Boolean Networks
Why Gamora?

**Graph Learning based solution:**
- Gate-level netlists are naturally represented as graphs
- Alternate solutions to recognition and classification
- Make better use to modern computing systems → better scalability
Why Gamora?

Successful examples using graph learning:

- Classify sub-circuit functionality from gate-level netlists
  - [TCAD’21] GNN-RE: Graph neural networks for reverse engineering of gate-level netlists

- Predict boundaries of arithmetic blocks
  - [ ICCAD’21] Graph learning-based arithmetic block identification

- Predict the functionality of approximate circuits

- Analyze impacts of circuit rewriting on functional operator detection
  - [ GLSVLSI’22] Graph neural network based netlist operator detection under circuit rewriting
How Gamora?

Symbolic reasoning requires structural and functional information from neighborhood nodes

- Reasoning can be formulated as node-level classification in graphs
- Message-passing mechanism in GNN computation:
  - Simultaneously handling structural and functional information from Boolean networks
  - Analogous to symbolic propagation and structural hashing
How Gamora?

Flattened gate-level netlist in And-Inverter Graph (AIG)

Conventional methods
- Structural hashing + functional propagation
- Low scalability
- Low parallelism
How Gamora?

Conventional methods
- Structural hashing + functional propagation
- Low scalability
- Low parallelism

Our solution: GNN

Flattened gate-level netlist in And-Inverter Graph (AIG)
How Gamora?

Node features to encode Boolean functional information

- Conventional methods
  - Structural hashing + functional propagation
  - Low scalability
  - Low parallelism

- Our solution: GNN
How Gamora?

Node features to encode Boolean functional information

Input edges complemented or not?

PI/PO or intermediate node? complemented or not?

Conventional methods
- Structural hashing + functional propagation
- Low scalability
- Low parallelism

Our solution: GNN
How Gamora?

Node features to encode Boolean functional information

Conventional methods
- Structural hashing + functional propagation
- Low scalability
- Low parallelism

Our solution: GNN
- Message-passing mechanism: functional and structural aggregation
How Gamora?

Node features to encode Boolean functional information

Conventional methods

- Structural hashing + functional propagation
- Low scalability
- Low parallelism

Our solution: GNN

- Message-passing mechanism: functional and structural aggregation
- Strong scalability
- Better utilization of modern compute platforms
How Gamora?

- How to handle large-scale Boolean networks?
  - GNN has better support from modern computing systems → GPU acceleration
  - Node-level, model-level, and graph-level parallelism → Billion-node scalability
Multiplier Verification

- Integer multipliers are ubiquitous components
  - Advanced multipliers, such as Booth multipliers, are difficult to verify.
  - [FMCAD’21] Sound and automated verification of real-world RTL multipliers

- Large multipliers are important in homomorphic encryption
  - [ISCAS’14] Practical homomorphic encryption: A survey
  - [CSUR’18] A survey on homomorphic encryption schemes: Theory and implementation
Multiplier Verification

Our goal: identify the adder tree in flattened/bit-blasted multiplier netlists

- Essential step in symbolic computer algebra for multiplier verification
Single Adder Extraction

- A full adder has a SUM and a CARRY
Single Adder Extraction

- A full adder has a SUM and a CARRY
- \( \text{SUM} = \text{XOR}_3(1,2,3) \)
Single Adder Extraction

- A full adder has a SUM and a CARRY
- $\text{SUM} = \text{XOR}3(1,2,3)$
Single Adder Extraction

- A full adder has a SUM and a CARRY
- SUM = XOR3(1,2,3)

(a) AIG
(b) OUT6 = XOR(1,2)
OUT9 = XOR(6,3)
(c) OUT9 = XOR3(1,2,3)
Single Adder Extraction

- A full adder has a SUM and a CARRY
- SUM = XOR3(1,2,3)
- CARRY = MAJ3(1,2,3)

(a) AIG
(b) OUT6 = XOR(1,2)
OUT9 = XOR(6,3)
(c) OUT9 = XOR3(1,2,3)
(d) AIG of a full adder
Adder Tree Extraction

Example: AIG of 3-bit multiplier
Adder Tree Extraction

Example: AIG of 3-bit multiplier

Step 1: find XOR nodes
Adder Tree Extraction

Example: AIG of 3-bit multiplier

Step 1: find XOR nodes

Step 2: find MAJ nodes
Adder Tree Extraction

Example: AIG of 3-bit multiplier

Step 1: find XOR nodes

Step 2: find MAJ nodes

Step 3: find boundary

Root

Nodes and connections diagram
Adder Tree Extraction

Example: AIG of 3-bit multiplier

Step 1: find XOR nodes

Step 2: find MAJ nodes

Step 3: find boundary
Adder Tree Extraction

Example: AIG of 3-bit multiplier

Step 1: find XOR nodes
- XOR

Step 2: find MAJ nodes
- MAJ

Step 3: find boundary
- Root

Can we simultaneously perform multiple steps?
Solution: Multi-task GNN

AIG

Structural info
AIG topology

Functional info
- PI/PO/intermediate nodes
- Whether each input edge is complemented

GraphSAGE

Task 1: XOR classification
Task 2: MAJ classification
Task 3: Root classification

Multi-label annotated AIG
Adder extraction

- XOR
- MAJ
- Root
Solution: Multi-task GNN

- **Structural info**: AIG topology
- **Functional info**: • PI/PO/intermediate nodes • Whether each input edge is complemented

---

**GraphSAGE**

- Fuse **structural** and **functional** information
- Neighbor sampling and aggregation to get node embeddings

\[
\begin{align*}
    h^k_{N(v)} &\leftarrow \text{AGGREGATE}_k(\{h^k_{u}, \forall u \in N(v)\}) \\
    h^k_v &\leftarrow \sigma(W^k \cdot \text{CONCAT}(h^k_{v}, h^k_{N(v)}))
\end{align*}
\]
Solution: Multi-task GNN

Multi-task learning
- **Knowledge sharing** across tasks to guarantee reasoning precision
- Improve **sample efficiency** during training
- Loss function:
  \[ \mathcal{L} = \alpha \cdot \ell(\overline{y_1}, y_1) + \beta \cdot \ell(\overline{y_2}, y_2) + \gamma \cdot \ell(\overline{y_3}, y_3) \]

Task 1: XOR classification
Task 2: MAJ classification
Task 3: Root classification
Experiment Setup

- **AIG-based multiplier netlists**
  - Carry-save-array (CSA) multipliers
  - Booth-encoded multipliers

- **Technology mapping**
  - The reduced standard-cell library mcnc.genlib (with gate input size <=3) from SIS distribution
  - ASAP 7nm technologies

- **Baseline & ground truth**
  - Logic synthesis tool ABC, using the adder tree extraction command
  - [TCAD’17] Fast algebraic rewriting based on and-inverter graphs
  - [TCAD’19] Understanding algebraic rewriting for arithmetic circuit verification: a bit-flow model

- Gamora is trained with **small** bitwidth multipliers (typically less than 32-bit) and evaluated on **large** bitwidth multipliers (up to 2048-bit)
Evaluation: CSA Multiplier

Sensitivity analysis on carry-save-array (CSA) multipliers

- Single- and multi-task
- With and without functional info
- Different training size (2-bit to 10-bit)
Evaluation: CSA Multiplier

Sensitivity analysis on carry-save-array (CSA) multipliers
- Single- and multi-task
- With and without functional info
- Different training size (2-bit to 10-bit)
Evaluation: Booth and Tech Mapping

CSA and Booth multipliers, with simple and complex technology mapping

- **Generalization** from small to large designs
- **Generalization** from before to after simple tech mapping
Evaluation: Runtime and Scalability

Runtime comparison between Gamora and ABC.

<table>
<thead>
<tr>
<th>Bitwidth of Multipliers</th>
<th>Runtime (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>ABC</td>
</tr>
<tr>
<td>128</td>
<td>Gamora</td>
</tr>
<tr>
<td>256</td>
<td></td>
</tr>
<tr>
<td>512</td>
<td></td>
</tr>
<tr>
<td>1024</td>
<td></td>
</tr>
</tbody>
</table>

6 orders of magnitude

ABC: a logic synthesis framework well adopted in academia


Further speedup with batched reasoning on a single GPU.
Summary

Gamora: a novel symbolic reasoning framework, which exploits GNNs to imitate structural hashing and functional aggregation in conventional reasoning approaches.

- **High reasoning performance** that reaches almost 100% and over 97% accuracy for CSA and Booth-encoded multipliers, which is still over 92% in finding functional modules after complex technology mapping;
- **Strong scalability** to Boolean networks with over 33 million nodes, with up to six orders of magnitude speedups compared to the state-of-the-art implementation in the ABC framework;
- **Great generalization capability** from simple to complex designs, such as from small to large bitwidth multipliers, and from before to after technology mapping.

Gamora reveals the great potential of applying GNNs and GPU acceleration to speed up symbolic reasoning, which is available at [https://github.com/Yu-Utah/Gamora](https://github.com/Yu-Utah/Gamora).
Thanks!