IBM IEEE CAS/EDS
AI Compute Symposium 2021
October 13-14

Program-to-Circuit:
Exploiting GNNs for Program Representation and Circuit Translation

Nan Wu[1], Huake He[1], Yuan Xie[1], Pan Li[2], Cong Hao[3]
nanwu@ucsb.edu
[1] *University of California, Santa Barbara*
[2] *Purdue University*
[3] *Georgia Institute of Technology*

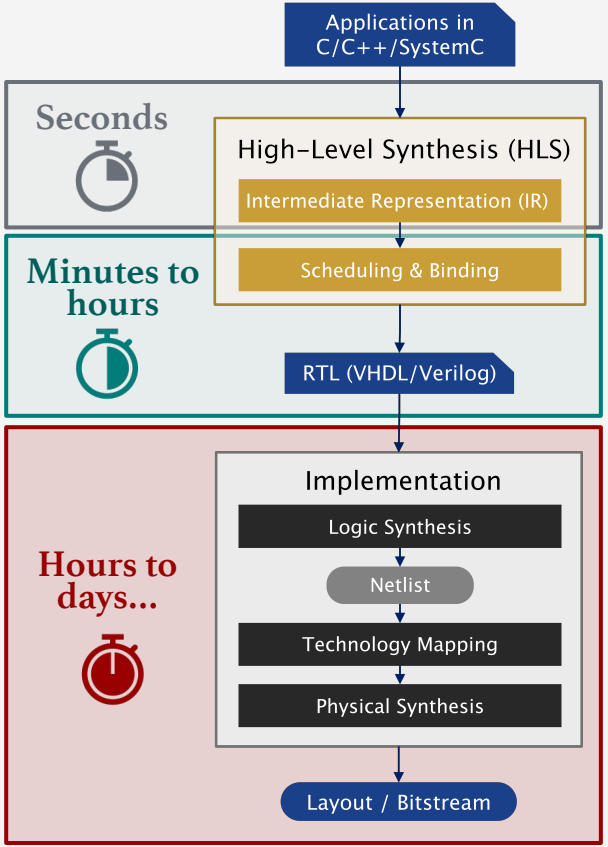
Motivation

Why Program-to-Circuit matters?

- Getting the actual circuit quality after implementation is very time-consuming → one major obstacle of *hardware agile development*
- Fast and accurate circuit quality evaluation from the earlier design stage, the better.

Program-to-Circuit:

- The **circuit evaluation** during the translation from behavioral languages to circuit designs, which expedites the evaluation process by orders of magnitude.



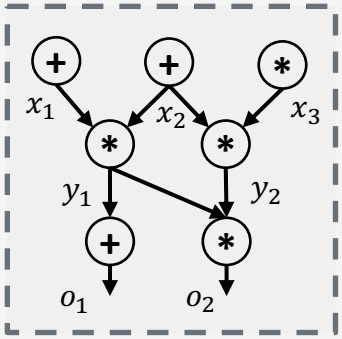
Problem Formulation

Graph Neural Networks

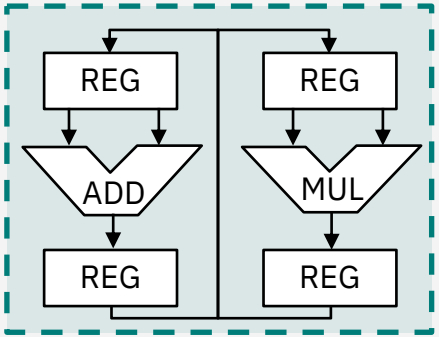
- Generalize to unseen designs
- Predict implementation quality fast and accurately

```
x1 = a + b;
x2 = b + c;
x3 = d * e;
y1 = x1 * x2;
y2 = x2 * x3;
o1 = y1 + 5;
o2 = y1 * y2;
```

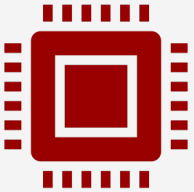
Seconds



Minutes to hours



Hours to days...



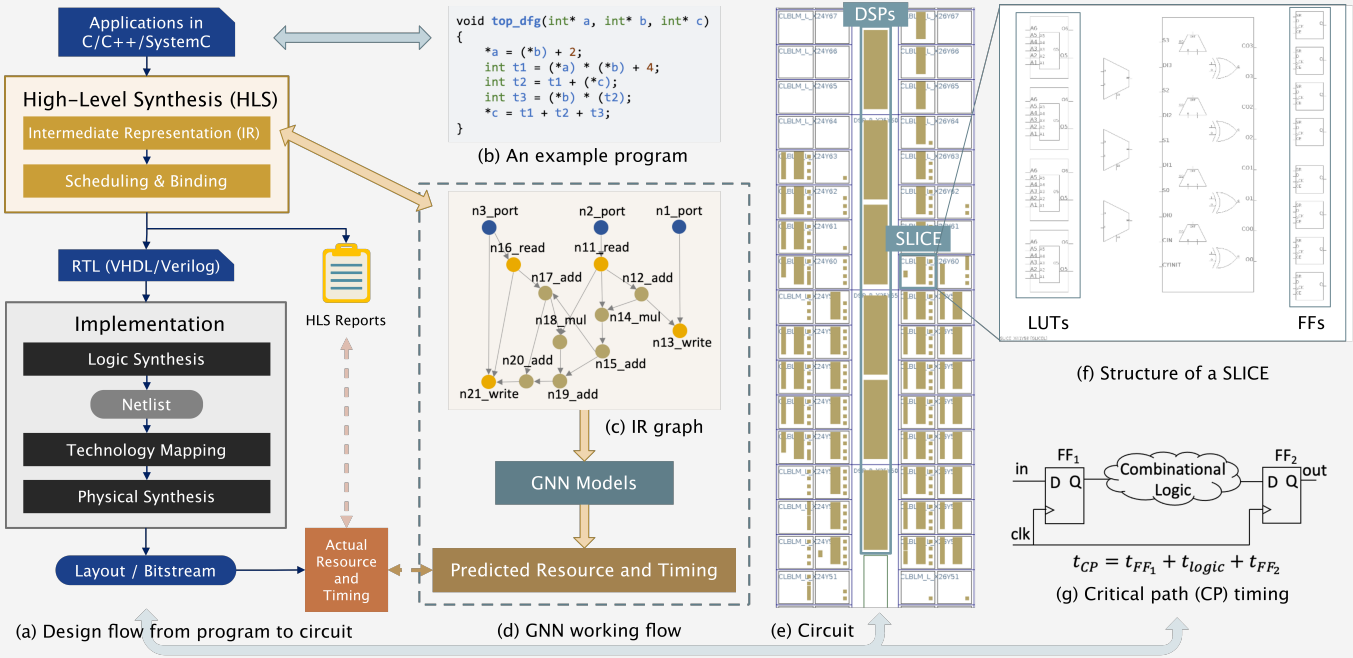
Program

Data Flow Graph (DFG)

RTL Design

Actual Implementation

Case Study Problem: FPGAs as the target circuit platform



(a) The design flow from behavioral programs to hardware circuits. (b) An example program written in C++. (c) The IR graph extracted by compiler front-ends. (d) The working flow of GNNs, predicting actual resource usage and timing merely based on raw IR graphs. (e) The translated circuit from the example program, using resources including DSPs, LUTs, FFs, and SLICEs. (f) The detailed structure of a SLICE. (g) An example of CP timing.

Implementation

Benchmarks

- **Synthetic programs**: 19,120 basic blocks, and 18,570 programs with branches.
- Three **real-world HLS benchmarks**: MachSuite, CHStone, and PolyBench
- Translated into circuit designs, with actual resource utilization and timing after implementation.

14 GNNs

- **Graph convolutional network (GCN) and variants**: (1) GCN; (2) GCN with a virtual node; (3) SGC, simplified version of GCN; (4) GraphSage; (5) GCN with auto-regressive moving average filters (ARMA); (6) GCN that considers path integral information.
- **Graph isomorphism network (GIN) variants**: (1) GIN; (2) GIN with a virtual node; (3) Principal neighborhood aggregation (PNA)
- **Employment of multi-relational information**: (1) Graph attention networks (GATs); (2) Gated graph neural networks (GGNNs); (3) Relational GCN (RGCN)
- **Inspired from vision tasks**: (1) Graph U-Net; (2) GNN with the feature-wise linear modulation (FiLM).
- Identifying key design challenges → to provide domain-specific knowledge for designing GNNs with suitable inductive biases.

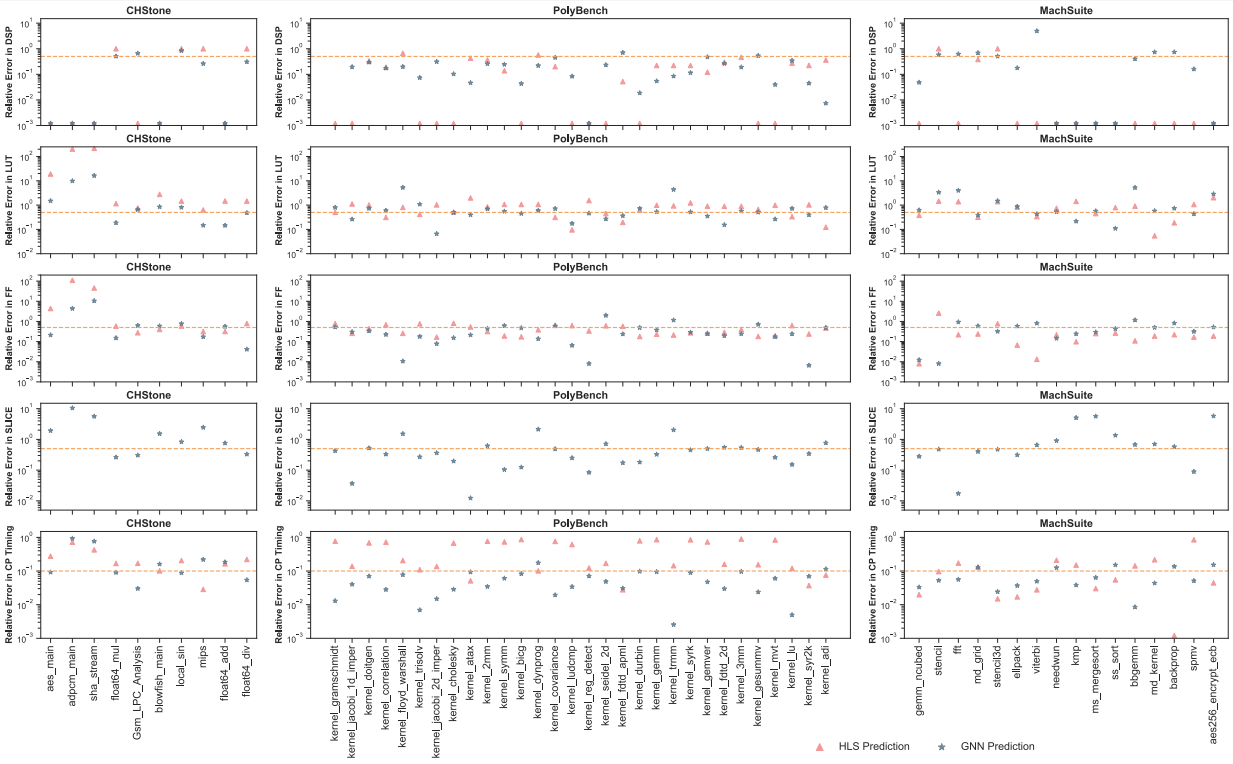
Results on Synthetic Programs

	DFG					CDFG				
	DSP	LUT	FF	SLICE	CP Timing	DSP	LUT	FF	SLICE	CP Timing
GCN	16.31%	16.49%	21.27%	22.29%	**6.12%**	25.30%	28.64%	38.34%	43.63%	8.79%
GCN - Virtual Node	15.72%	15.93%	21.64%	23.21%	6.36%	17.31%	33.93%	39.94%	49.22%	**8.13%**
SGC	42.12%	23.93%	30.61%	28.20%	7.92%	44.01%	60.87%	53.50%	64.10%	10.32%
GraphSage	15.18%	14.01%	17.11%	15.90%	**6.12%**	17.01%	28.09%	39.11%	46.53%	**8.25%**
ARMA	19.12%	13.46%	16.87%	16.09%	6.50%	18.47%	**25.21%**	32.15%	28.31%	8.42%
PAN	15.24%	14.13%	17.23%	16.49%	6.38%	16.88%	32.65%	44.36%	44.84%	8.54%
GIN	15.52%	16.10%	22.08%	22.63%	6.58%	15.47%	28.48%	38.82%	46.12%	8.76%
GIN - Virtual Node	15.04%	16.17%	23.09%	24.19%	6.40%	17.94%	29.40%	48.64%	49.44%	8.59%
PNA	**12.65%**	**11.64%**	**14.41%**	**14.34%**	6.26%	**14.71%**	**22.86%**	**26.47%**	**23.38%**	8.87%
GAT	26.22%	22.64%	27.74%	25.68%	8.30%	28.66%	46.19%	54.73%	64.72%	10.32%
GGNN	15.40%	13.64%	16.94%	15.79%	6.47%	16.28%	28.05%	31.88%	34.15%	8.50%
RGCN	**13.27%**	13.03%	**15.09%**	**15.11%**	6.14%	**15.03%**	26.33%	**25.52%**	**22.84%**	8.72%
UNet	18.40%	14.90%	19.17%	17.18%	6.61%	18.92%	32.83%	53.06%	54.35%	9.02%
FiLM	20.05%	**12.50%**	16.94%	16.30%	6.27%	17.42%	26.97%	27.35%	31.50%	8.67%

Two Takeaways:

- The relational information (i.e., edge information) is important in IR graphs, since they represent data dependency, control dependency, or a mix of these two, which is a critical basis in logic synthesis and impacts resource allocation.
- Equipped with multiple aggregators, PNA is more powerful to characterize different neighborhood information of each node, thus making better predictions.

Results on Real-world Benchmarks



GNN predictions

- Slightly better in resource
- Obviously better in critical path timing, which is hard-to-predict.

Challenges posed by real-world benchmarks:

- Generalization across a wider range of graph sizes.
- Discrimination across graphs with heavy loops.
- Compatibility with more complex operations in applications.
- Capability to handle out-of-distribution cases.

HLS Prediction **GNN Prediction**

Potential Directions & Contributions

Three potential directions that would benefit the Program-to-Circuit problem.

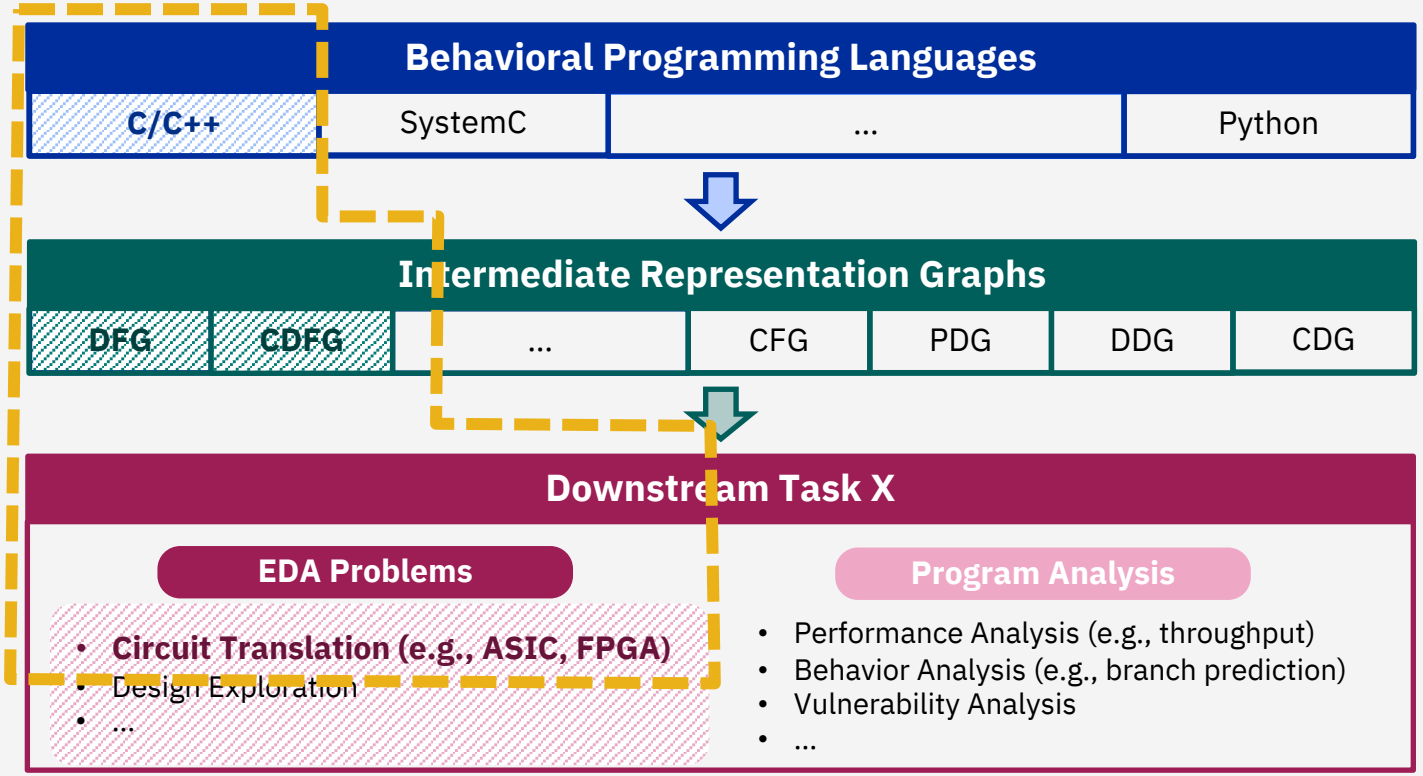
- Generalization across **graph sizes, node degrees, and out-of-distribution cases**.
- Structural and algorithmic innovations to process **heavy-loop graph topologies**.
- Gap from node/link classification to **graph-level regression**.

Contribution

- **Problem definition** of Program-to-Circuit.
- **Benchmark** consisting of synthetic and real programs.
- **SOTA GNN analysis** from domain-specific aspects.
- **Real-case generalization analysis** to provide pointers for future GNN development along this promising direction.

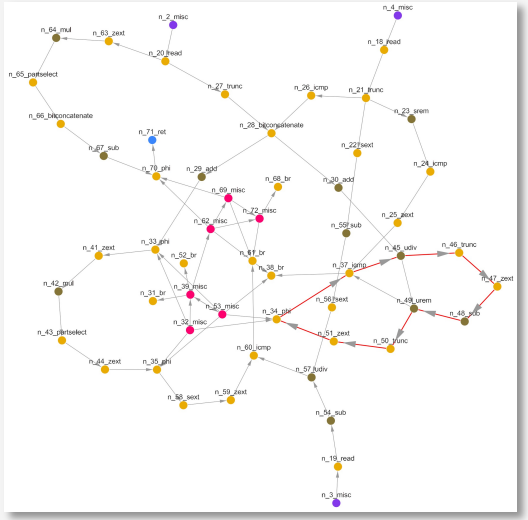
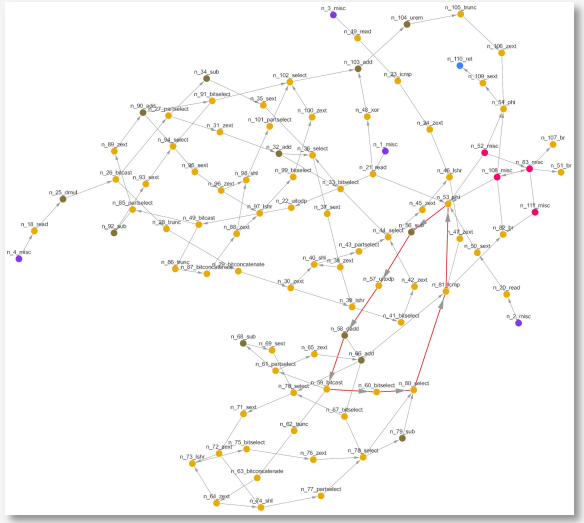
Broader Impact

From **Program-to-Circuit** to **Program-to-X**



More Information

Full text

- https://arxiv.org/abs/2109.06265

More Visualizations of CDFGs