MONTH 20XX

TEXAS
The University of Texas at Austin

PREFIXRL

Using RL to Optimize Parallel Prefix Circuits
Work done By: R. Roy et al. @ NVIDIA

PRESENTED BY: KAYVAN MANSOORSHAHI
The University of Texas at Austin

Outline

- Prefix sum
- PrefixRL
  - Goals
  - Problems + solutions
- Results

Scan

- For input **X,** and operator *, Scan results in **Y** defined as:

$$y_i = x_i * x_{i-1} * x_{i-2} * \ldots * x_0 \text{ for } 0 \leq i < N$$

- If we take * as addition we get prefix sum

Prefix Sum Example

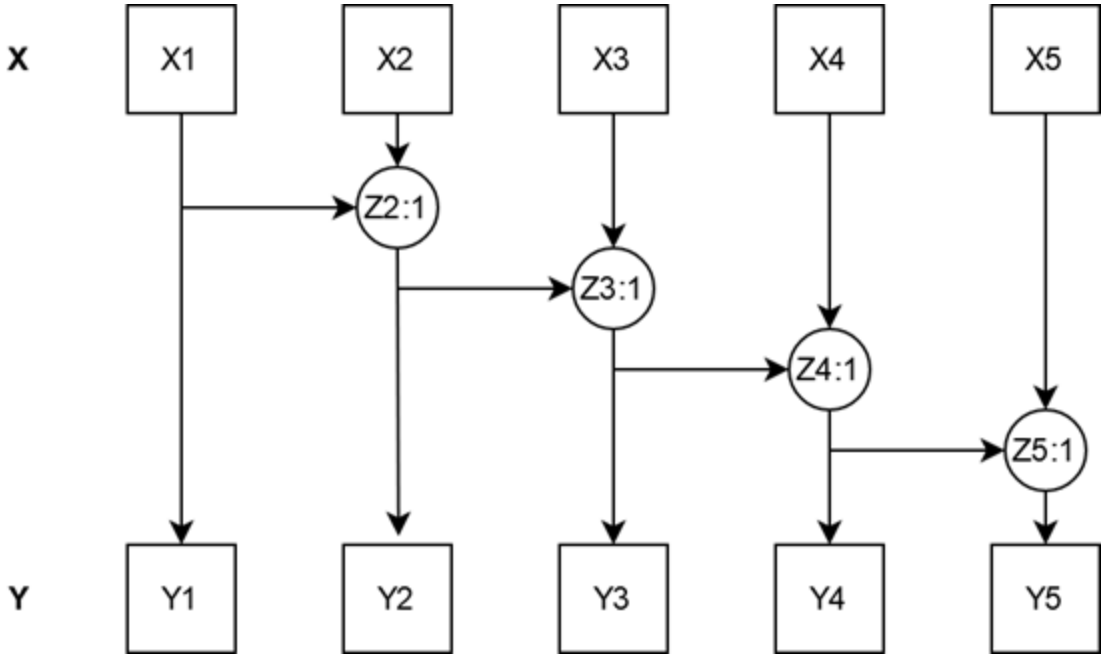
- **X** = [1,2,3,4]
- **Y** = [1, 1+2, 1+2+3, 1+2+3+4]

- Really easy sequentially
  - **Y**[i] = **Y**[i-1] + **X**[i] for 2 <= i <= |**X**|
  - **Y**[1] = **X**[1]

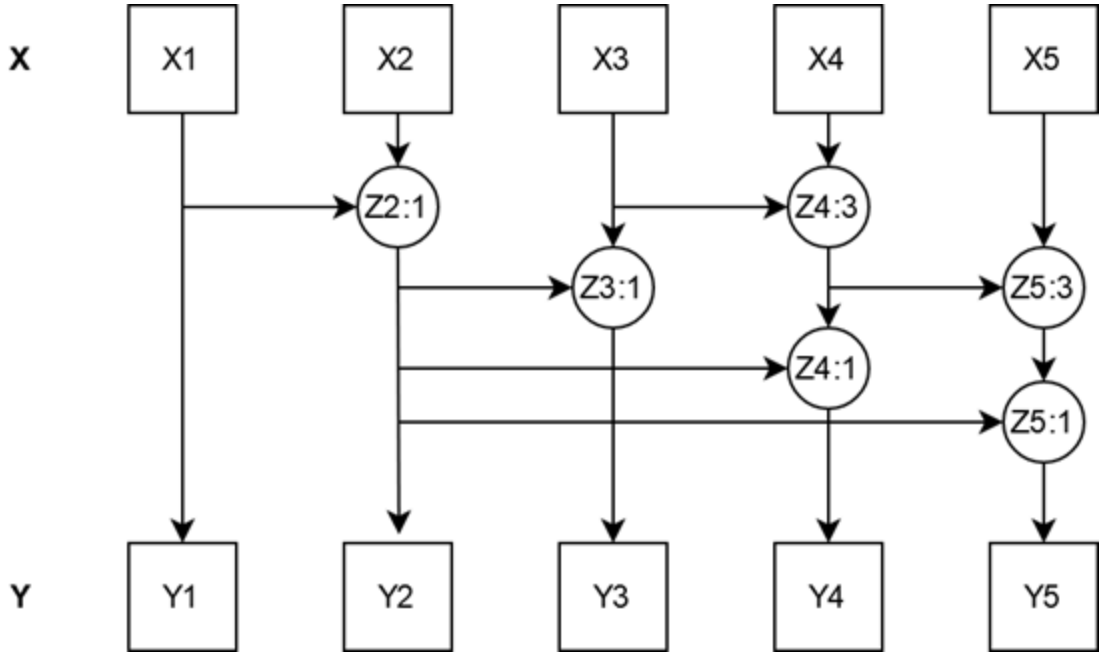
Prefix Sum

- Since * is associative we can divide and conquer, and reuse results
  - We will call these intermediate results $z_{j:i} = x_j * \ldots * x_i$
- Inputs (**X**) can be thought of as intermediate product $z_{i:I}$
- Outputs (**Y**) can be thought of as intermediate product $z_{i:0}$
- All $z_{i:j}$ can be split into $z_{i:k}$, and $z_{k-1:j}$
  - $z_{i:k}$ is referred to as the “upper parent” and $z_{k-1:j}$ as the “lower parent”

Prefix Sum Graphs



Prefix Sum Graphs



Adders as Prefix sum

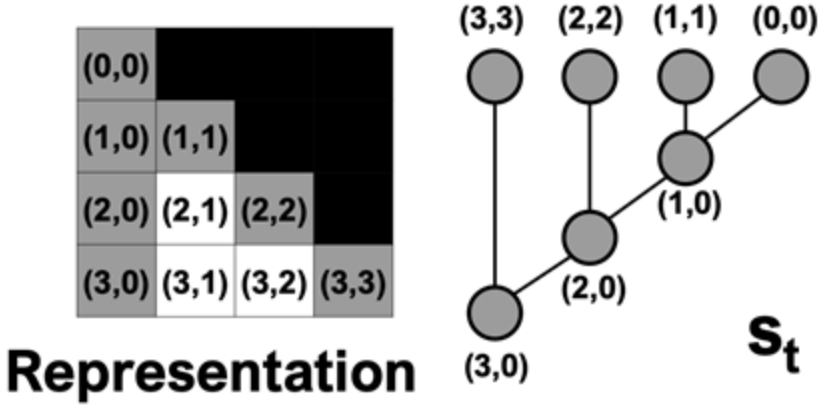
- Adders are ubiquitous circuits. Optimizing PPA is crucial
- Can be expressed as a prefix sum of bits
  - Can also be parallelized to improve performance
- SOTA has been Kogge-Stone adder since 1973. Can we do better?

- State space is huge ($O(2^{N\hat{}2})$), need to explore intelligently
- PPA of a circuit is expensive to collect (~36 sec per circuit)

PrefixRL

- Large unknown space? RL!

- State space is all LEGAL circuits
  - A matrix: (i,j) means generating $z_{i:j}$
- Action is add or remove (i,j)
- Value is <area, delay>
  - Getting reward is expensive
  - More than one reward



Credit: [1]

Legal Circuits Only

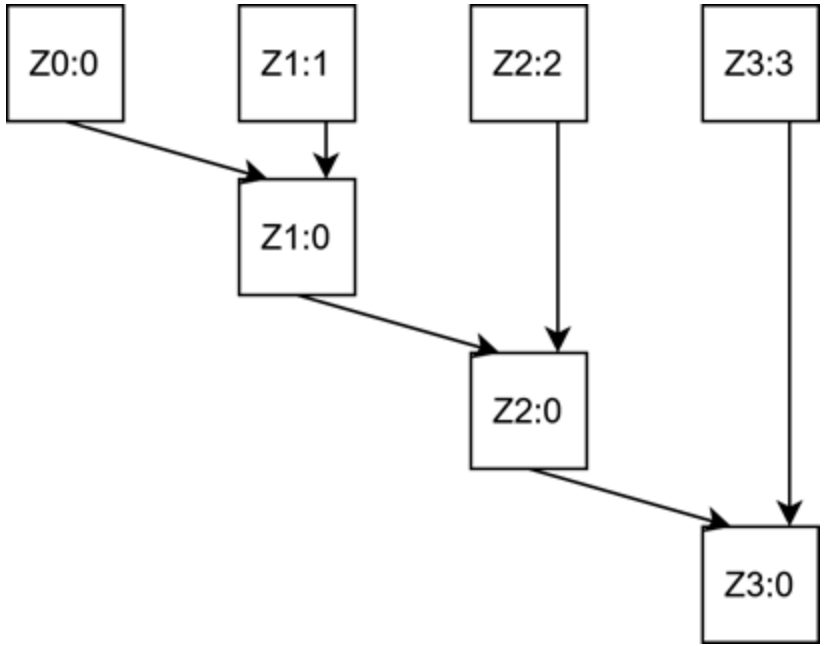
- Adding arbitrary intermediate can result in invalid circuit

- Add a legalization step before evaluating the new state
- For each node (MSB_{node}, LSB_{node}):
  - Upper parent = get node (MSB_{node}, max)
  - Lower parent = find node ($LSB_{Upper\ parent}$-1, LSB_{node})

- Upper parent will always exist since starting adder is legal
- Lower parent will be added by legalizer if it doesn't exist

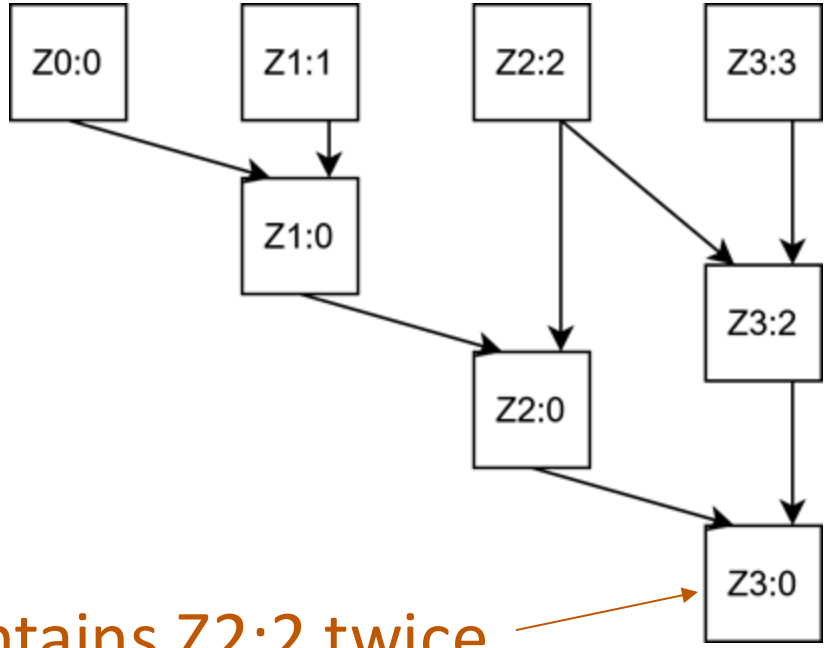
Legal Circuits Only

- Starting state
- Sequential RCA



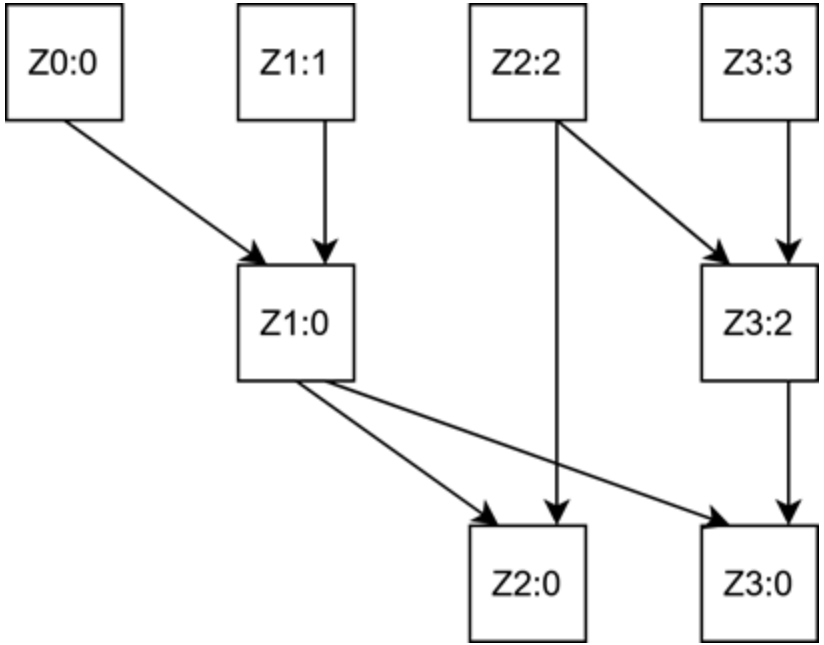
Legal Circuits Only

- Agent says add 3,2
- Parents are picked
  - (3:3) is upper parent
  - (2:2) is lower parent
- Adder is now invalid



Legal Circuits Only

- Visit 3,0 node
- Apply rules to validate
- Legal again



Legalizer

- Minlist: non-lower parent nodes
- Nodelist: all nodes
- Adding existing nodes is banned
- Delete can only touch min list
  - Legalizer will reverse it otherwise

Algorithm 1: PrefixRL N-input prefix graph actions

```
Function Initialize:
    nodelist ← ∅, minlist ← ∅;
    for m ← 0 to (N − 1) do                    // add in/out nodes
        add (m, m), (m, 0) to nodelist
    end
Function Add(msb,lsb):
    add (msb, lsb) to minlist;
    // remove new node's and child's lps from minlist
    for l ← (msb − 1) to 0 do
        if (msb, l) is in minlist then
            delete lp(msb, l) from minlist
        end
    end
    Legalize()
Function Delete(msb,lsb):
    delete (msb, lsb) from minlist;
    Legalize()
Function Legalize:
    nodelist ⟵ minlist;
    for m ← 0 to (N − 1) do                    // add in/out nodes
        add (m, m), (m, 0) to nodelist
    end
    for m ← (N − 1) to 0 do                    // add missing lps
        for l ← (m − 1) to 0 do
            if (m, l) is in nodelist then
                add lp(m, l) to nodelist
            end
        end
    end
```

Credit: [1]

Multiple Objectives

- Both area and delay are important
- Often at odds with each other
- Simple Norm won't work
- Scalarized deep Q-learning

$$\mathbf{r_t} = [area(s_t) - area(s_{t+1}), delay(s_t) - delay(s_{t+1})]$$

$$\mathbf{y_t} = \mathbf{r_t} + \gamma \mathbf{Q}(s_{t+1}, \underset{a}{\text{argmax}}[\mathbf{w}^\top \mathbf{Q}(s_{t+1}, a; \theta_t)]; \theta'_t)$$

$$Loss(\mathbf{Q}(s_t, a_t; \theta_t), \mathbf{y_t})$$

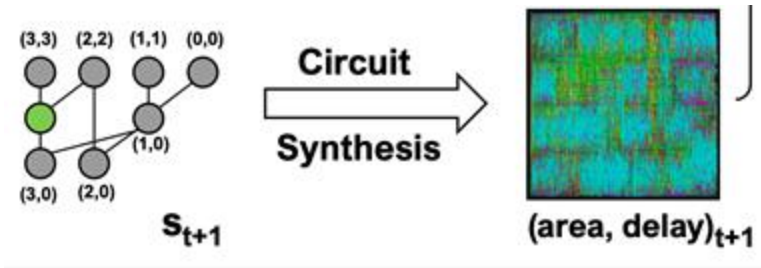
$$a_t = \underset{a}{\text{argmax}}[\mathbf{w}^\top \mathbf{Q}(s_t, a; \theta_t)]$$

- Network learns each separately
- **w** used for action selection
- **w** picked for objective preference

Credit: [1]

Circuit Synthesis in the Loop

- What is the area and delay of a graph?
- Could count nodes
  - Doesn't account for optimizations
  - Doesn't account for route
  - Doesn't account for fanout



- Synthesis in the loop
- Circuit synthesis is slow

Credit: [1]

Circuit Synthesis in the Loop

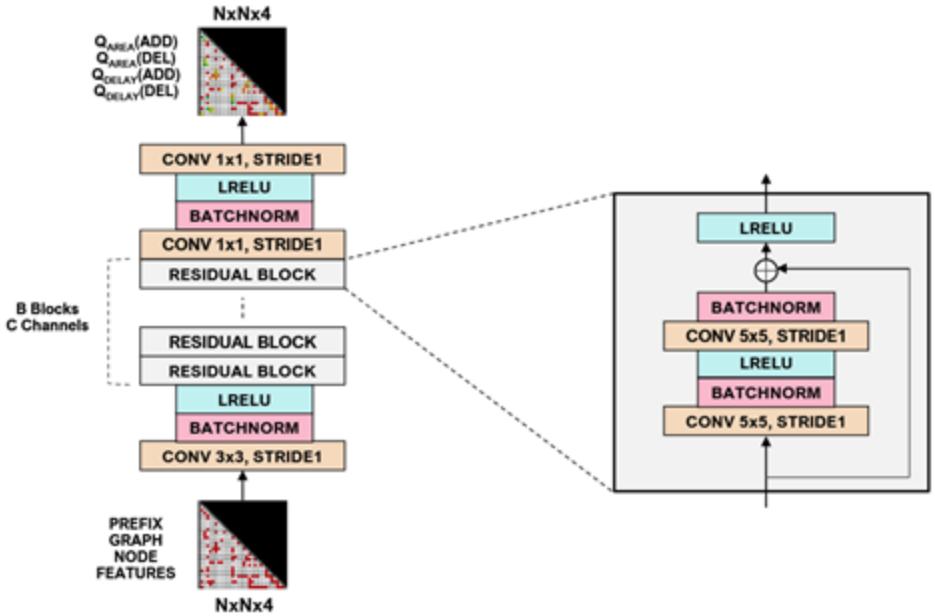
- Synth is long running, independent work
- Distribute, collect results, and learn

- They use Deep Q network (DQN)
- DQN is off policy, can use samples after Θ changes
- DQN can separate sample collection and training
  - Need to read more. Don’t fully understand why this is the case
- picked 192 workers to cover latency

Q-Network Architecture

- Q network shown left
- Inputs are NxNx4
  - node list
  - min list
  - level/depth of node
  - fanout of node

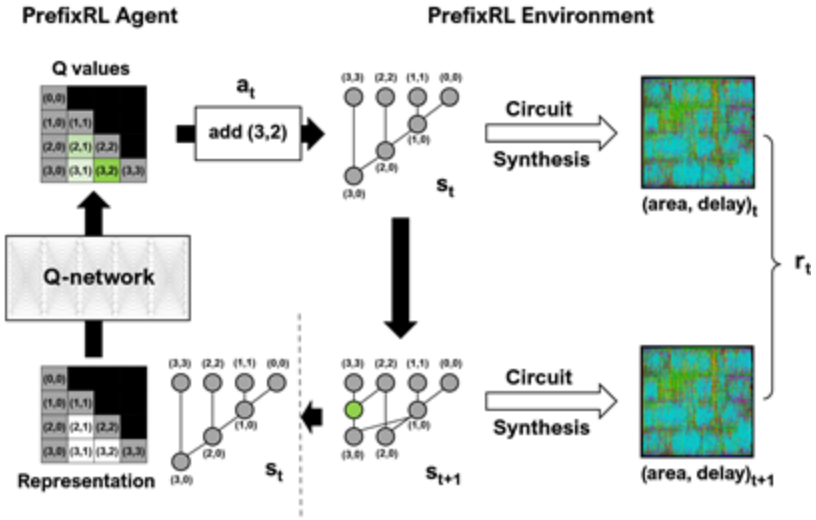
- Outputs are NxNx4
  - Q_{Area} for add/delete
  - Q_{Delay} for add/delete



Credit: [1]

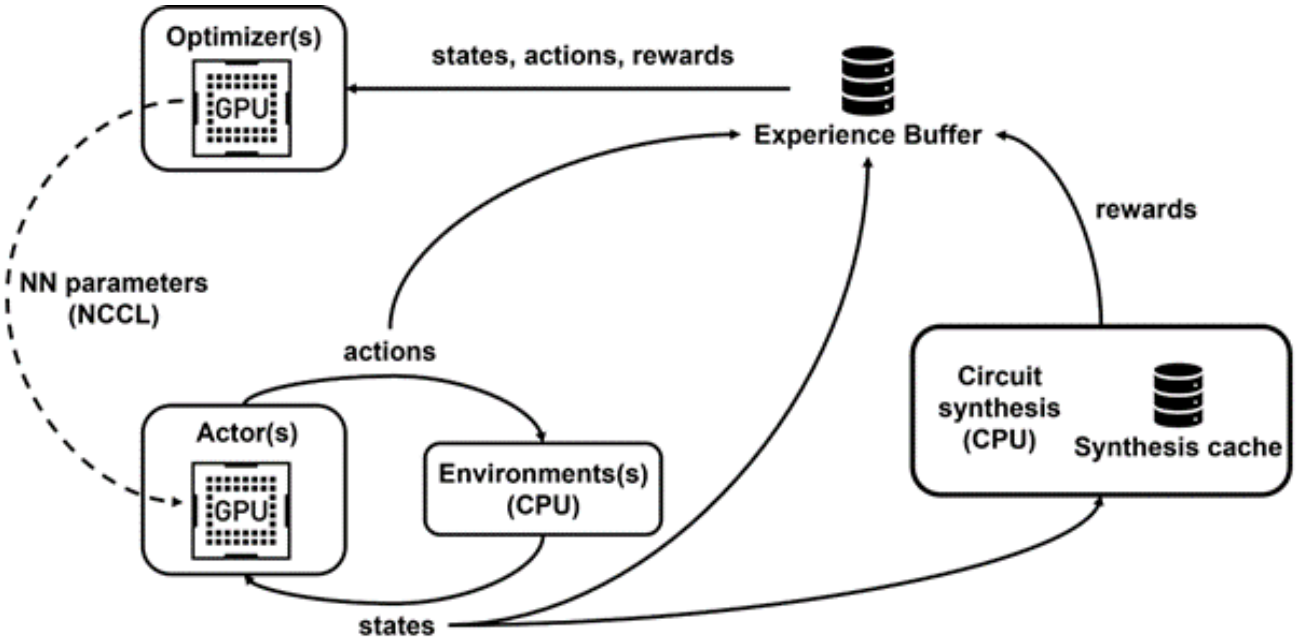
Training

- Aren't specific, but my best guess
- Main loop will
  - Query Q values of Actions using Θ
  - Act
  - Legalize
  - Ship to synth farm
  - Update Θ' using sample from buffer
  - After 60 steps, copy Θ' to Θ
- The synth farm will
  - Return synthesis result to replay buffer
- Again, need to understand DQN



Credit: [1]

Training



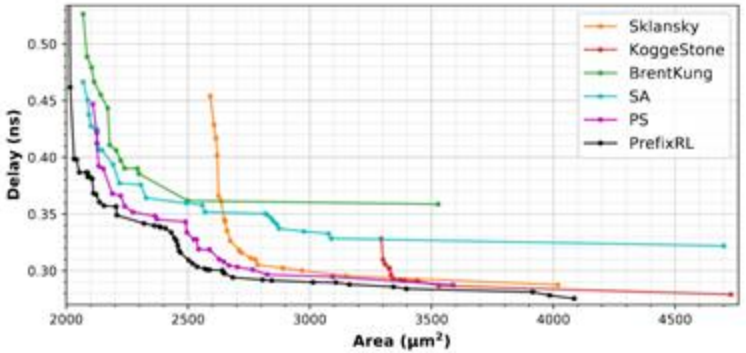
Credit: Nvidia Developer Blog https://developer.nvidia.com/blog/designing-arithmetic-circuits-with-deep-reinforcement-learning/

Results

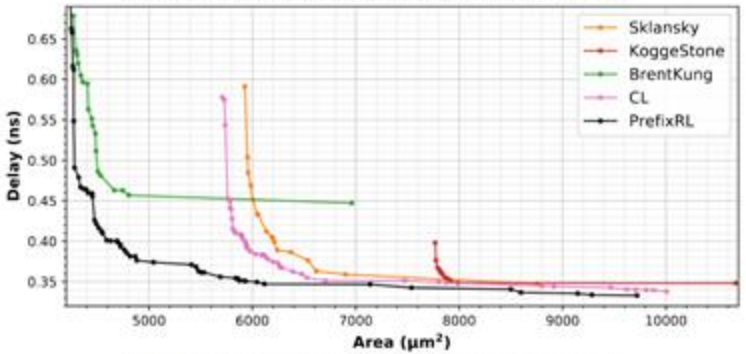
- Sklansky, BrentKung, KoggeStone are all SOTA human results
  - “Sit down and think”

- SA is simulated annealing
- PS is pruned search
- CL is machine learning

- 32b saves max 16% area
- 64b saves max 30% area



(a) 32b Adder Synthesis. OpenPhySyn, Nangate45.



(b) 64b Adder Synthesis. OpenPhySyn, Nangate45.

Credit: [1]

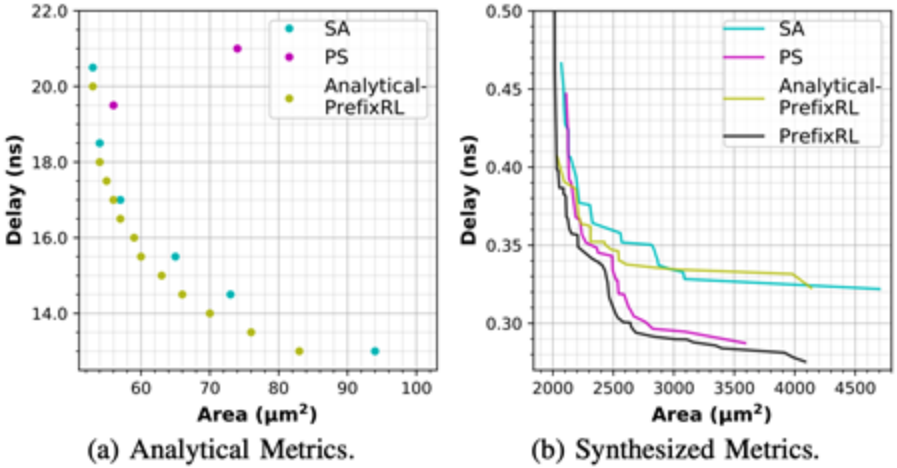
Conclusions

- RL is effective for design space exploration

- Analytical steps can reduce the space (Legalization)
- Can learn complex post processing steps (Synthesis)
- Can amortize cost of expensive rewards (Distributed training)
- When learning multiple objectives
  - scalarize for action selection
  - otherwise keep them separate

Results mod Synthesis

- RL dominates with heuristics too
  - RL beats SA with its own heuristics
- These heuristics don't translate
  - Still RL search the space "better"



(a) Analytical Metrics. (b) Synthesized Metrics.

Credit: [1]

Extras

- Reward of area and delay are different units, so can't be compared
  - They simply scale each by some factor. I guess selected empirically
  - They scale area by .001 and delay by 10
- Synth results change based on delay targets
  - Synth at 4 delay targets, interpolate, and return optimal based on **w** vector
- Synth results are cached
  - If exploration factor is low the hit rate is 10%;50% for 64b;32b respectively

Extras

- How much did synthesis in the loop really buy?
  - Purely analytical method beats Kogge-Stone adder
  - Worse than synthesis in the loop (11% vs 30% reduction in area)
- Reinforcement learning agent can be used to search design spaces
- Expensive reward functions can be parallelized if off-policy method

Credits

[1] Roy, Rajarshi & Raiman, Jonathan & Kant, Neel & Elkin, Ilyas & Kirby, Robert & Siu, Michael & Oberman, Stuart & Godil, Saad & Catanzaro, Bryan. (2021). PrefixRL: Optimization of Parallel Prefix Circuits using Deep Reinforcement Learning. 853-858. 10.1109/DAC18074.2021.9586094. // (https://arxiv.org/pdf/2205.07000)