RL-MUL: Multiplier Design Optimization with Deep Reinforcement Learning

Dongsheng Zuo **Yikang Ouyang** **Yuzhe Ma**

December 21, 2024

The Hong Kong University of Science and Technology (Guangzhou)

Background and Motivation

- Multiplication is a fundamental operation in many applications.

Background and Motivation

- Multiplication is a fundamental operation in many applications.
- Multipliers are widely adopted in various circuits, especially in the AI field.

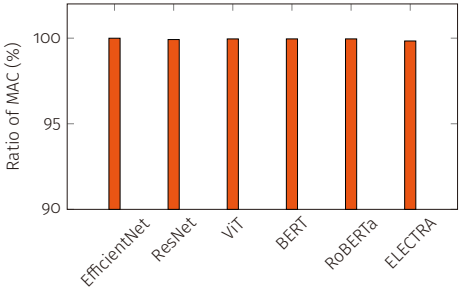


Figure 1: Ratios of MAC computations in various neural networks.

Background and Motivation

- Optimizing multipliers is challenging and non-trivial due to the huge design space.

C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Transactions on Electronic Computers*, vol. EC-13, no. 1, pp. 14–17, 1964.

L. Dadda, "Some schemes for fast serial input multipliers," in *1983 IEEE 6th Symposium on Computer Arithmetic (ARITH)*, 1983, pp. 52–59.

W. Xiao *et al.*, "Gomil: Global optimization of multiplier by integer linear programming,", pp. 374–379, 2021.

Background and Motivation

- Optimizing multipliers is challenging and non-trivial due to the huge design space.
- Previous multiplier design and optimization approaches:
  - *Manual design*: Wallace Tree, Dadda Tree, full custom multiplier

C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Transactions on Electronic Computers*, vol. EC-13, no. 1, pp. 14–17, 1964.

L. Dadda, "Some schemes for fast serial input multipliers," in *1983 IEEE 6th Symposium on Computer Arithmetic (ARITH)*, 1983, pp. 52–59.

W. Xiao *et al.*, "Gomil: Global optimization of multiplier by integer linear programming,", pp. 374–379, 2021.

Background and Motivation

- Optimizing multipliers is challenging and non-trivial due to the huge design space.
- Previous multiplier design and optimization approaches:
  - *Manual design*: Wallace Tree, Dadda Tree, full custom multiplier
  - *Mathematical programming*: GOMIL(ILP based)

C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Transactions on Electronic Computers*, vol. EC-13, no. 1, pp. 14–17, 1964.

L. Dadda, "Some schemes for fast serial input multipliers," in *1983 IEEE 6th Symposium on Computer Arithmetic (ARITH)*, 1983, pp. 52–59.

W. Xiao *et al.*, "Gomil: Global optimization of multiplier by integer linear programming,", pp. 374–379, 2021.

Background and Motivation

- Reinforcement learning (RL) enables efficient search in huge solution spaces.
- RL can make use of the actual evaluation within the optimization loop.
- PrefixRL use RL to optimize prefix adders

R. Roy *et al.*, "Prefixrl: Optimization of parallel prefix circuits using deep reinforcement learning,", 2021, pp. 853–858. DOI: 10.1109/DAC18074.2021.9586094.

Background and Motivation

- Reinforcement learning (RL) enables efficient search in huge solution spaces.
- RL can make use of the actual evaluation within the optimization loop.
- PrefixRL use RL to optimize prefix adders
- How to formulate multiplier optimization problem into an RL formulation?

R. Roy *et al.*, "Prefixrl: Optimization of parallel prefix circuits using deep reinforcement learning,", 2021, pp. 853–858. DOI: 10.1109/DAC18074.2021.9586094.

Background and Motivation

- Reinforcement learning (RL) enables efficient search in huge solution spaces.
- RL can make use of the actual evaluation within the optimization loop.
- PrefixRL use RL to optimize prefix adders
- How to formulate multiplier optimization problem into an RL formulation?
- We propose RL-MUL, a multiplier optimization framework based on deep reinforcement learning.

R. Roy *et al.*, "Prefixrl: Optimization of parallel prefix circuits using deep reinforcement learning,", 2021, pp. 853–858. DOI: 10.1109/DAC18074.2021.9586094.

Background and Motivation

- Reinforcement learning (RL) enables efficient search in huge solution spaces.
- RL can make use of the actual evaluation within the optimization loop.
- PrefixRL use RL to optimize prefix adders
- How to formulate multiplier optimization problem into an RL formulation?
- We propose RL-MUL, a multiplier optimization framework based on deep reinforcement learning.

R. Roy *et al.*, "Prefixrl: Optimization of parallel prefix circuits using deep reinforcement learning,", 2021, pp. 853–858. DOI: 10.1109/DAC18074.2021.9586094.

Background and Motivation

- Reinforcement learning (RL) enables efficient search in huge solution spaces.
- RL can make use of the actual evaluation within the optimization loop.
- PrefixRL use RL to optimize prefix adders
- How to formulate multiplier optimization problem into an RL formulation?
- We propose RL-MUL, a multiplier optimization framework based on deep reinforcement learning.

R. Roy *et al.*, "Prefixrl: Optimization of parallel prefix circuits using deep reinforcement learning,", 2021, pp. 853–858. DOI: 10.1109/DAC18074.2021.9586094.

Preliminaries

- The multiplier is usually implemented as three main parts:

Preliminaries

- The multiplier is usually implemented as three main parts:
  - A partial product generator (PPG)
  - A compressor tree (CT), which is the most critical part.
  - A carry propagation adder

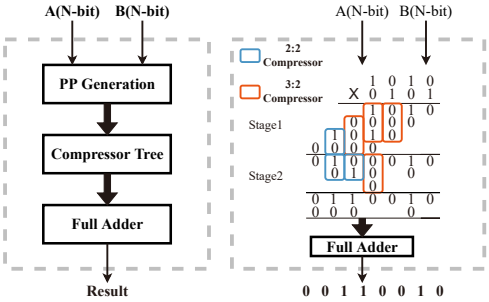


Figure 2: Multiplier architecture

Compressor Tree

- Build a compressor tree.

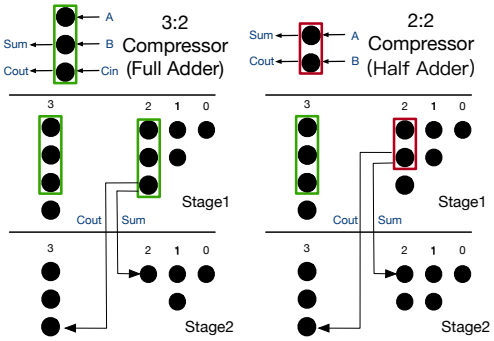


Figure 3: 3:2 and 2:2 compressor.

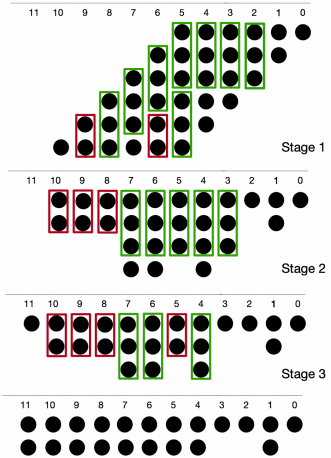
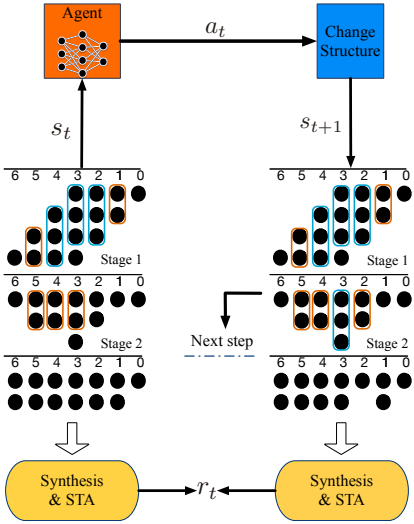


Figure 4: Compressor tree.

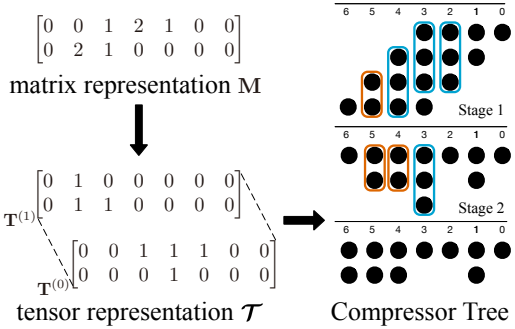
RL Framework

- Deep Q-learning(DQN) based framework
- ResNet-18 as the agent
- A state s refers to a structure.
- An action a refers to modification on current structure s
- Pareto-driven Reward



Multiplier Representation and State Space

- State space $\mathcal{S}$: all N-bit multiplier structures.
- We use a matrix $\mathbf{M} \in \mathbb{R}^{2N \times K}$ to present the structure.
  - Only consider total compressor number in each column.
- We also denote $\mathcal{T} \in \mathbb{R}^{K \times 2N \times ST}$ as the tensor representation
  - With compressor information of each stage and each column



Action Space

- Four actions for each column:
  - Add a 2:2 compressor
  - Delete a 2:2 compressor
  - Replace a 3:2 compressor with a 2:2 compressor
  - Replace a 2:2 compressor with a 3:2 compressor

Action Space

- Four actions for each column:
  - Add a 2:2 compressor
  - Delete a 2:2 compressor
  - Replace a 3:2 compressor with a 2:2 compressor
  - Replace a 2:2 compressor with a 3:2 compressor
- Hence, the action space $|\mathcal{A}| = 2N \times 4 = 8N$.

Action Space

- Four actions for each column:
  - Add a 2:2 compressor
  - Delete a 2:2 compressor
  - Replace a 3:2 compressor with a 2:2 compressor
  - Replace a 2:2 compressor with a 3:2 compressor
- Hence, the action space $|\mathcal{A}| = 2N \times 4 = 8N$.
- For a compressor tree with $2N$ columns, the output of a deep Q-network is a vector that indicates the predicted Q-values:

$$Q(s_t) = [q_{11}, q_{12}, q_{13}, q_{14}, \cdots, q_{2N,1}, q_{2N,2}, q_{2N,3}, q_{2N,4}], \tag{1}$$

Action Space

- Four actions for each column:
  - Add a 2:2 compressor
  - Delete a 2:2 compressor
  - Replace a 3:2 compressor with a 2:2 compressor
  - Replace a 2:2 compressor with a 3:2 compressor
- Hence, the action space $|\mathcal{A}| = 2N \times 4 = 8N$.
- For a compressor tree with $2N$ columns, the output of a deep Q-network is a vector that indicates the predicted Q-values:

$$Q(s_t) = [q_{11}, q_{12}, q_{13}, q_{14}, \cdots, q_{2N,1}, q_{2N,2}, q_{2N,3}, q_{2N,4}], \tag{1}$$

- Only legal actions can be selected.

Action Space

- Four actions for each column:
  - Add a 2:2 compressor
  - Delete a 2:2 compressor
  - Replace a 3:2 compressor with a 2:2 compressor
  - Replace a 2:2 compressor with a 3:2 compressor
- Hence, the action space $|\mathcal{A}| = 2N \times 4 = 8N$.
- For a compressor tree with $2N$ columns, the output of a deep Q-network is a vector that indicates the predicted Q-values:

$$Q(s_t) = [q_{11}, q_{12}, q_{13}, q_{14}, \cdots, q_{2N,1}, q_{2N,2}, q_{2N,3}, q_{2N,4}], \tag{1}$$

- Only legal actions can be selected.
- The compressor tree may not be legal after action.
  - We proposed an legalization strategy to refine the structure after action a, to ensure legality of the new structure s_{t+1}.

Pareto-driven Reward

- The total cost of a structure is calculated by a weighted sum of area and delay values in different constraints:

$$cost = \sum_{i=1}^{n} area_i + w \sum_{i=1}^{n} delay_i$$

where $area_i$ and $delay_i$ is the synthesised metrics with i-th constraint, w is the weight to trade off area and delay.

- We define our reward r as the difference between s_t and s_{t+1}:

$$r_t = cost_t - cost_{t+1}$$

R. Roy *et al.*, "Prefixrl: Optimization of parallel prefix circuits using deep reinforcement learning,", 2021, pp. 853–858. DOI: 10.1109/DAC18074.2021.9586094.

Experiment Settings

- Multiplier RTL Generator: EasyMAC
- *Synthesis tool*: Yosys built in OpenROAD
- *STA tool*: OpenSTA[Par] built in OpenROAD
- *Standard Cell Library*: NanGate 45nm Open Cell Library
- *Multiplier*: 8-bit and 16-bit, Booth and Non-booth

J. Zhang *et al.*, "Easymac: Design exploration-enabled multiplier-accumulator generator using a canonical architectural representation,", 2022, pp. 647–653.

Experiment Settings

- Multiplier RTL Generator: EasyMAC
- *Synthesis tool*: Yosys built in OpenROAD
- *STA tool*: OpenSTA[Par] built in OpenROAD
- *Standard Cell Library*: NanGate 45nm Open Cell Library
- *Multiplier*: 8-bit and 16-bit, Booth and Non-booth
- *Baselines*: GOMIL (ILP), Wallace, simulated annealing (SA)

J. Zhang *et al.*, "Easymac: Design exploration-enabled multiplier-accumulator generator using a canonical architectural representation,", 2022, pp. 647–653.

Experiment Settings

- Multiplier RTL Generator: EasyMAC
- *Synthesis tool*: Yosys built in OpenROAD
- *STA tool*: OpenSTA[Par] built in OpenROAD
- *Standard Cell Library*: NanGate 45nm Open Cell Library
- *Multiplier*: 8-bit and 16-bit, Booth and Non-booth

- *Baselines*: GOMIL (ILP), Wallace, simulated annealing (SA)
- *RL training*:
  - 50 episodes
  - 60 steps in each episode
  - *Runtime*: About 20s per step for 16-bit multipliers.

J. Zhang *et al.*, "Easymac: Design exploration-enabled multiplier-accumulator generator using a canonical architectural representation,", 2022, pp. 647–653.

Experimental Results

- We run synthesis flow for multipliers from RL-MUL and baseline approaches:

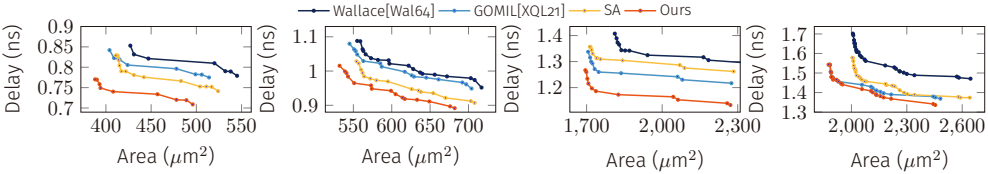


Figure 5: Pareto-frontiers of the synthesis results on multipliers. From left to right: 8-bit AND-based; 8-bit MBE-based; 16-bit AND-based; 16-bit MBE-based

Experimental Results

- We further validated by implemented multipliers from RL-MUL and baseline approaches in processing element (PE) arrays.

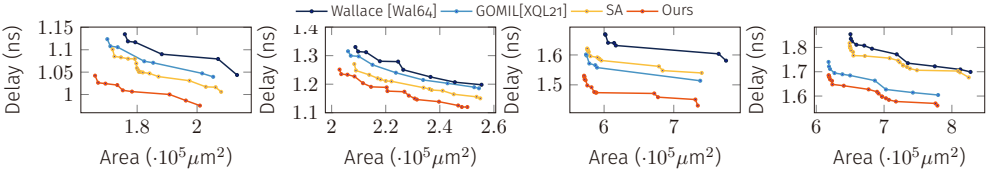


Figure 6: Pareto-frontiers of the synthesis results on PE arrays. From left to right: 8-bit AND-based; 8-bit MBE-based; 16-bit AND-based; 16-bit MBE-based

Thanks for listening