Georgia Tech School of Electrical and Computer Engineering
College of Engineering

SYNERGY
http://synergy.ece.gatech.edu

ConfuciuX: Hardware Design-space Exploration via RL and Optimization

Sheng-Chun (Felix) Kao
Georgia Institute of Technology

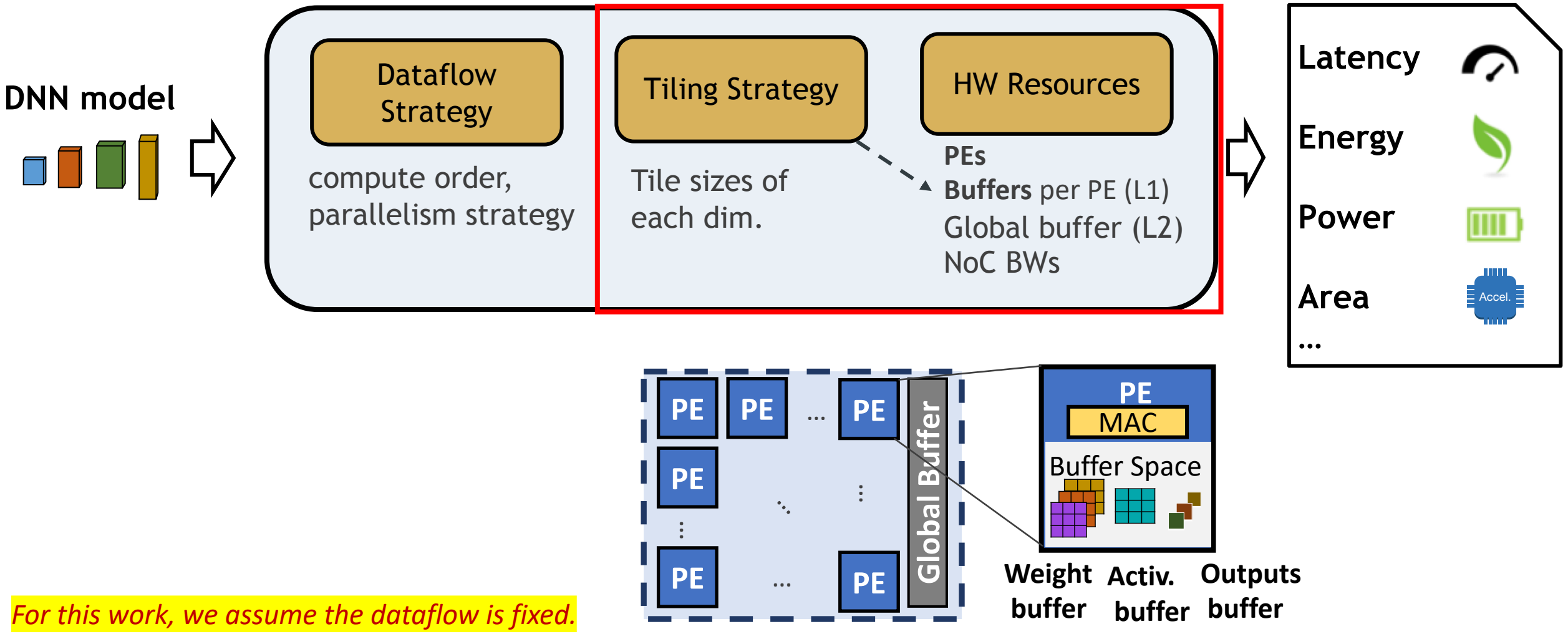
MICRO 2020

Date: Oct. 17, 2020

S.-C. Kao, G Jeong, T Krishna, "ConfuciuX: Autonomous Hardware Resource Assignment for DNN Accelerators using Reinforcement Learning", MICRO, 2020

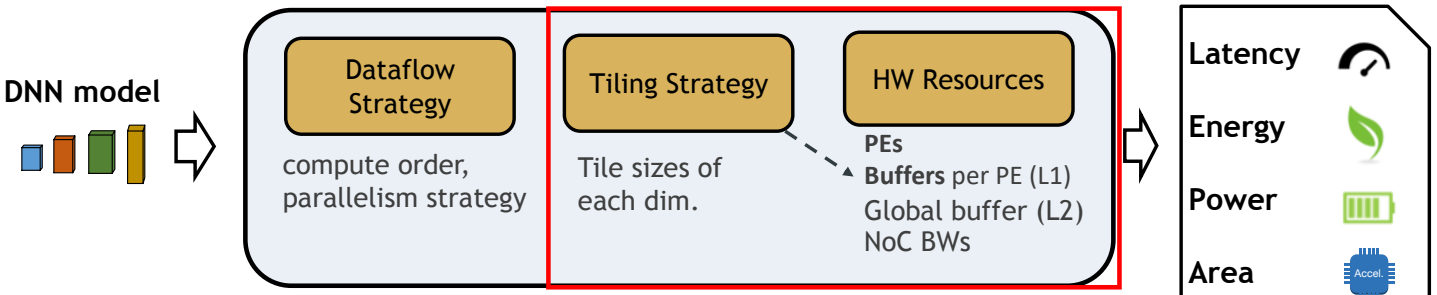
Code available: https://github.com/maestro-project/confuciux

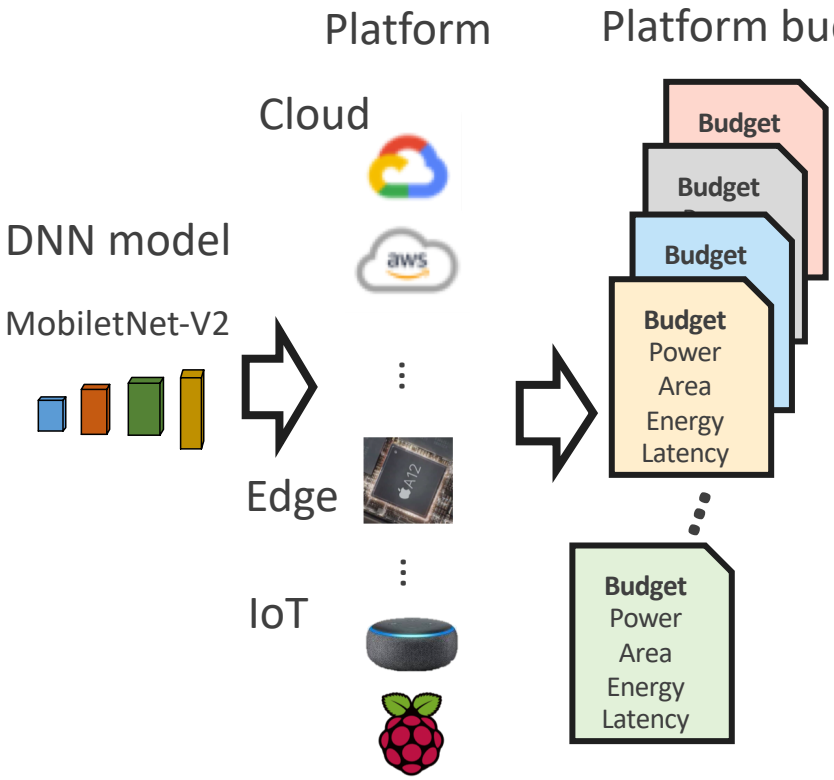
Architecture of DNN Accelerator

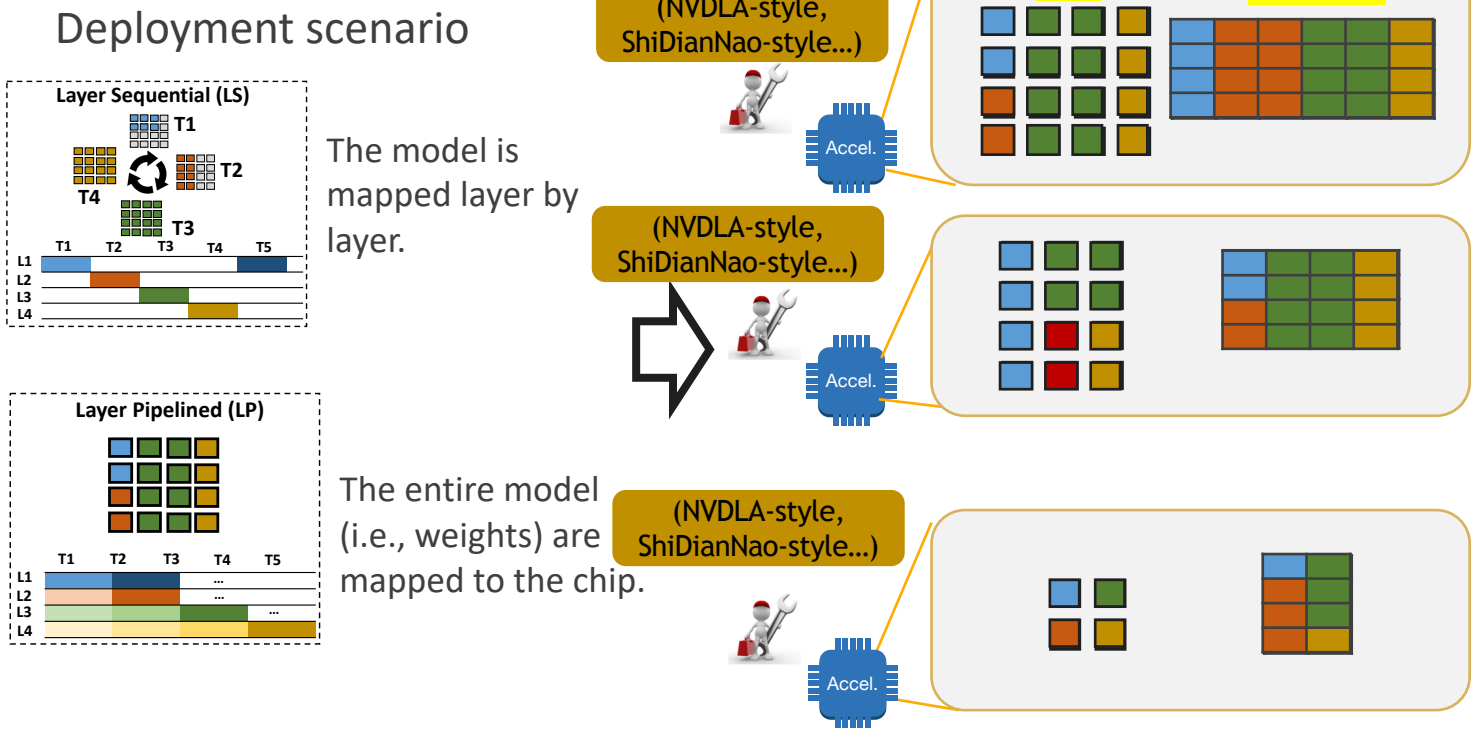


For this work, we assume the dataflow is fixed.

Architecture of DNN Accelerator

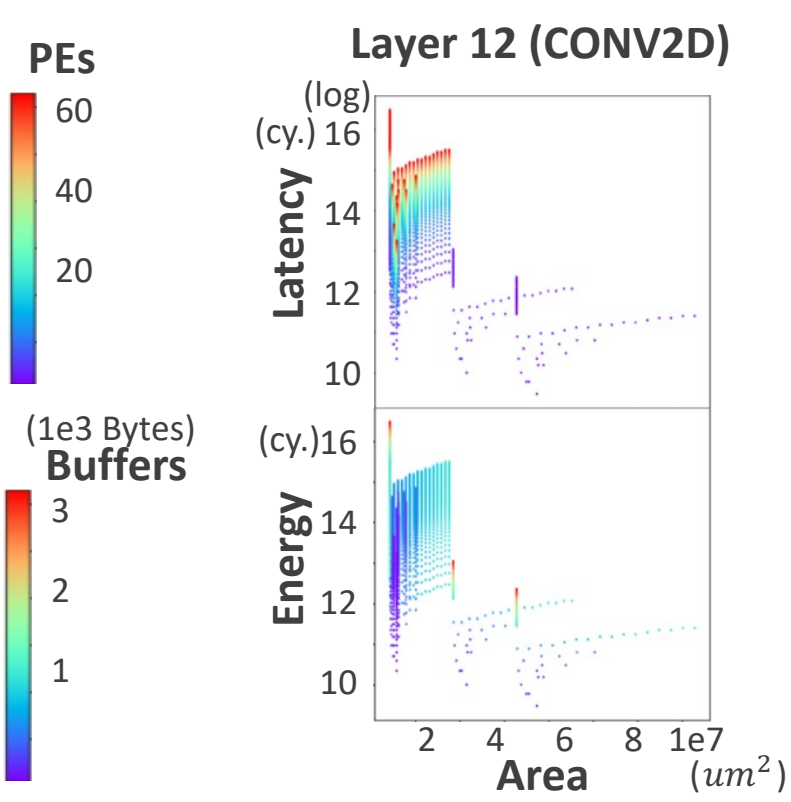


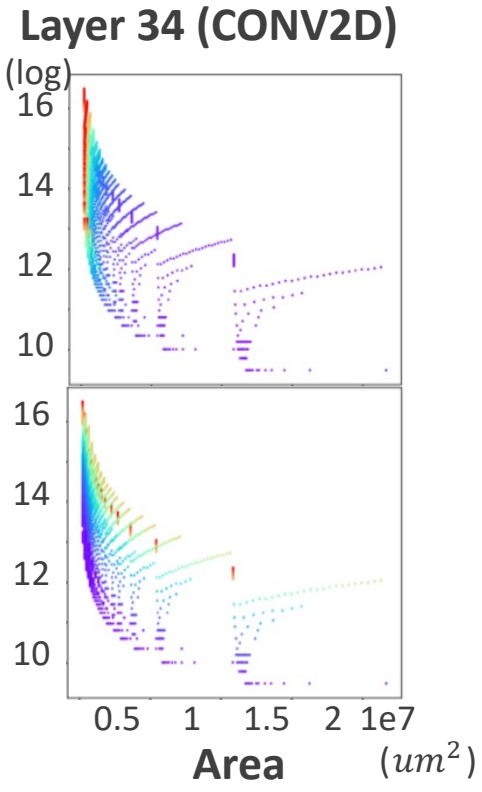


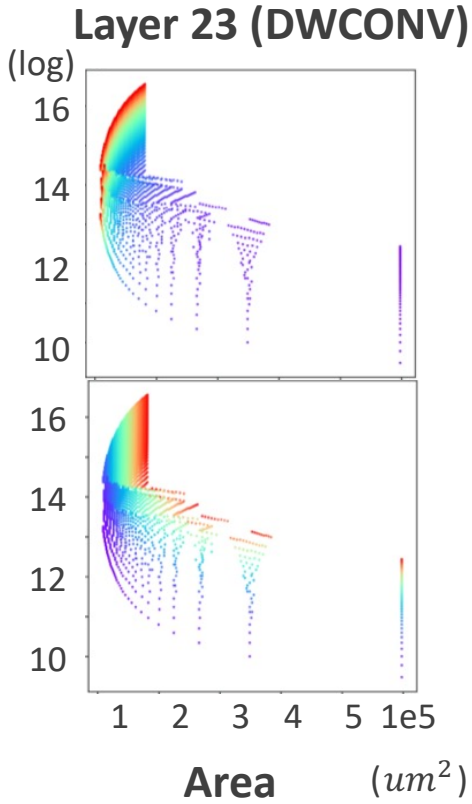


What is the impact of HW resources on performance and energy?

The Impact of HW Resources Assignment







Selected layer from MobileNet-V2

Different layers react to the PEs/ Buffers differently.

https://github.com/maestro-project/maestro

Casting HW Resource Assignment as RL

Goal: Given power/area budget, find {PE, Buf} for each layer in a layer pipelined deployment scenario, such that objective (e.g., latency) is minimized

RL algorithm:

- Multiple episodes (until the algorithm converges)

Each episode:

- Multiple steps (until the environment terminates)
- Environment terminates when entire DNN is run or if environment runs out of area or power

Episode ➔ series of decision for each layer in a model (i.e., entire DNN model)

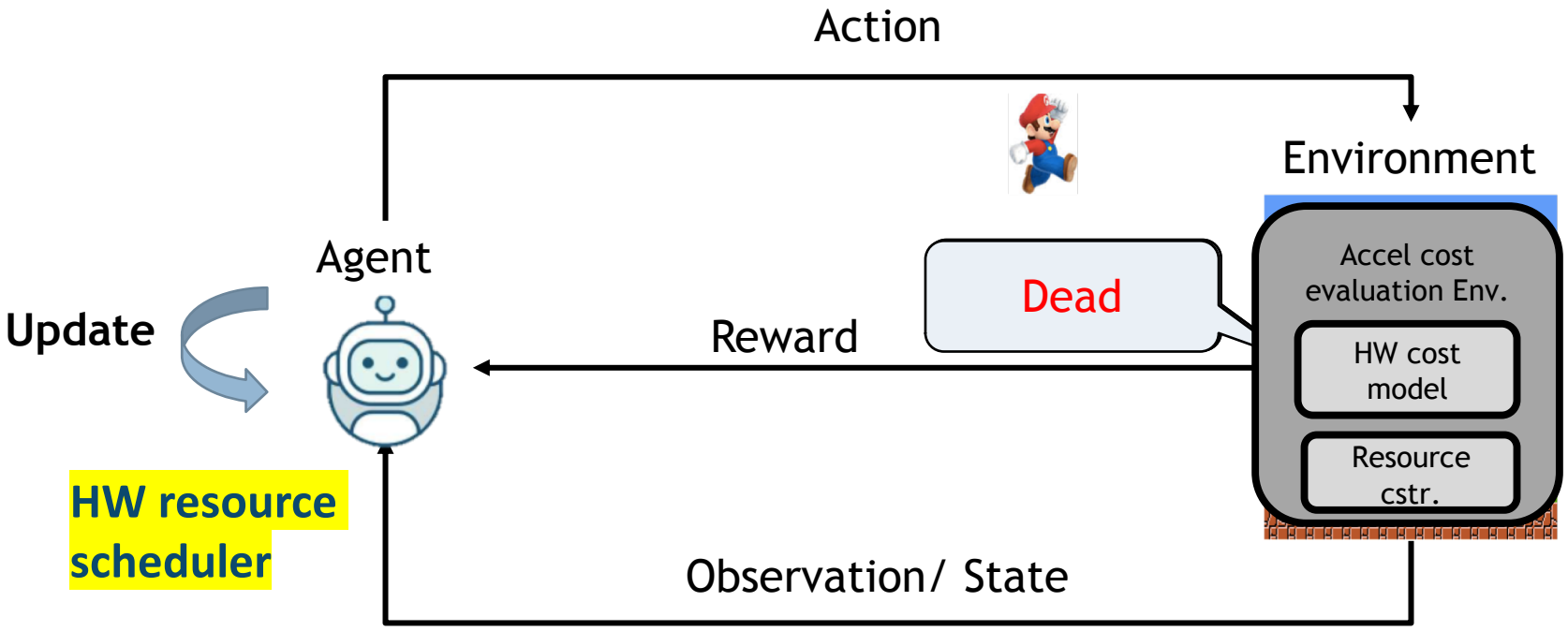
Each step:

- Agent assigns {PE, Buf} to a layer
- Environment tracks the remaining power/area budget

Step ➔ single DNN layer

Layer Pipelined: The entire model (i.e., weights) are mapped to the chip.

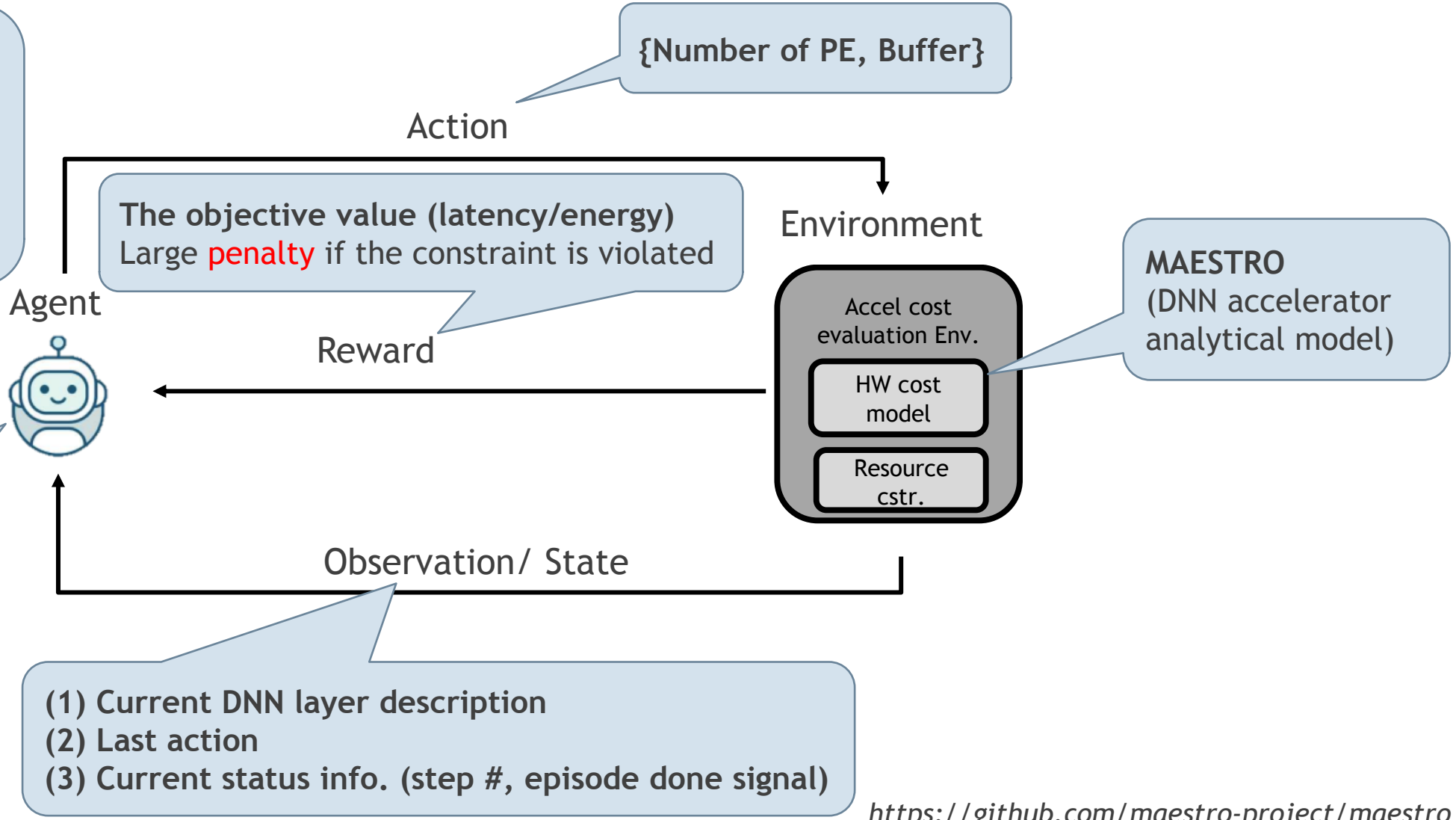
Template of Reinforcement Learning



Algorithm Details

Objective:
Latency or energy

Platform constraint:
Total chip area or power



https://github.com/maestro-project/maestro

Action Space

The fine-grained search/action space is extreme large

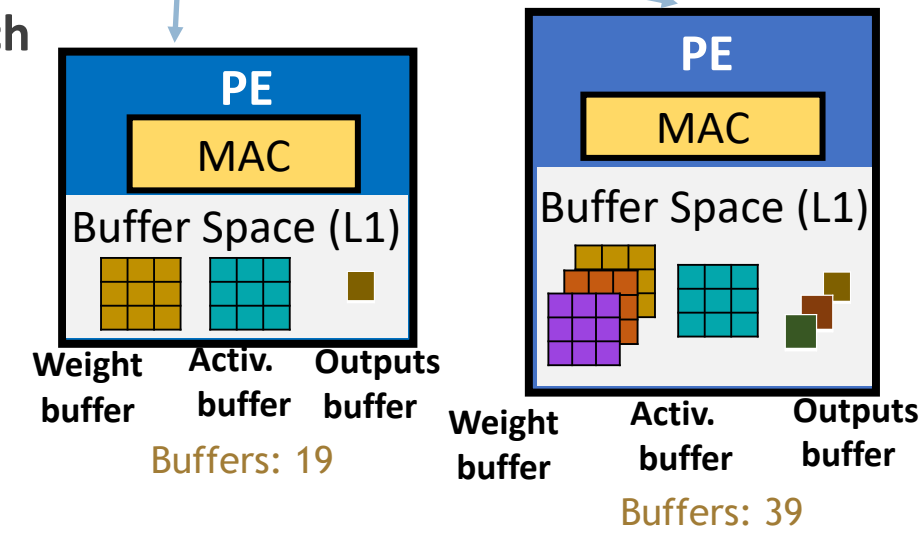
- 52 layers MobileNet-V2: 2^{728} → $O(10^{211})$ years

	Action level values
PEs	1, 2, 4, 8, 12, 16, 24, 32, 48, 64, 96, 128
Buffers (e.g., NVDLA-style)	19, 29, 39, 49, 59, 69, 79, 89, 99, 109, 119, 129

Approach: Two-stage optimization

- **Stage 1: Reinforment Learning (RL) coarse-grained global search**
  - We limit the actions of {PE}, {Buf} to 12 different values
    - PE sizes chosen empirically
    - Buf sizes increase by the unit of tile sizes

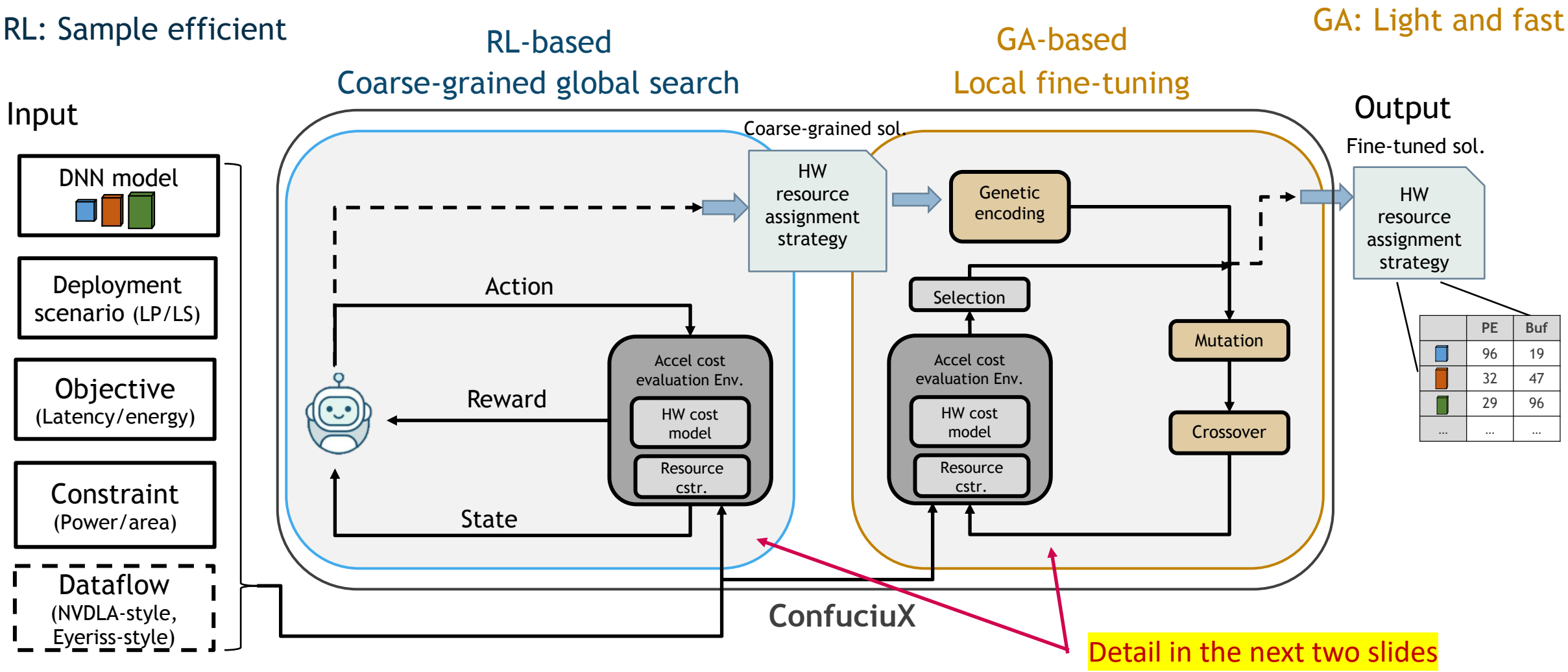
- **Stage 2: Genetic Algorithm (GA) local fine-tuning**
  - Fine-tune the stage-1 sol. locally



Overview of ConfuciuX

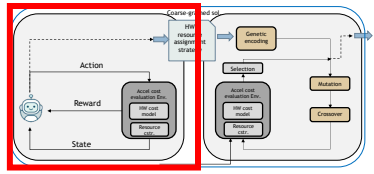
RL: Sample efficient

GA: Light and fast



More detail: S.-C. Kao, G Jeong, T Krishna, "ConfuciuX: Autonomous Hardware Resource Assignment for DNN Accelerators using Reinforcement Learning", MICRO'53, 2020

Algorithm Flow of ConfuciuX RL-based Search



Layer Pipelined Deployment

Episode 1

PE	Buf
4	6

Action

Reward: -100

State

PE	Buf
9	7

Accel cost evaluation Env.
- HW cost model
- Resource cstr.

Accum. reward

Episode

Output HW assignment strategy

	PE	Buf	Reward
	12	4	32
	9	7	13
	4	6	-100

Accumulated Reward : -55

Constraint is checked at every step, if violated, it early terminate with large penalty.

Episode 2

PE	Buf
5	3

Action

Reward: 15

State

PE	Buf
12	9

Accel cost evaluation Env.
- HW cost model
- Resource cstr.

Output HW assignment strategy

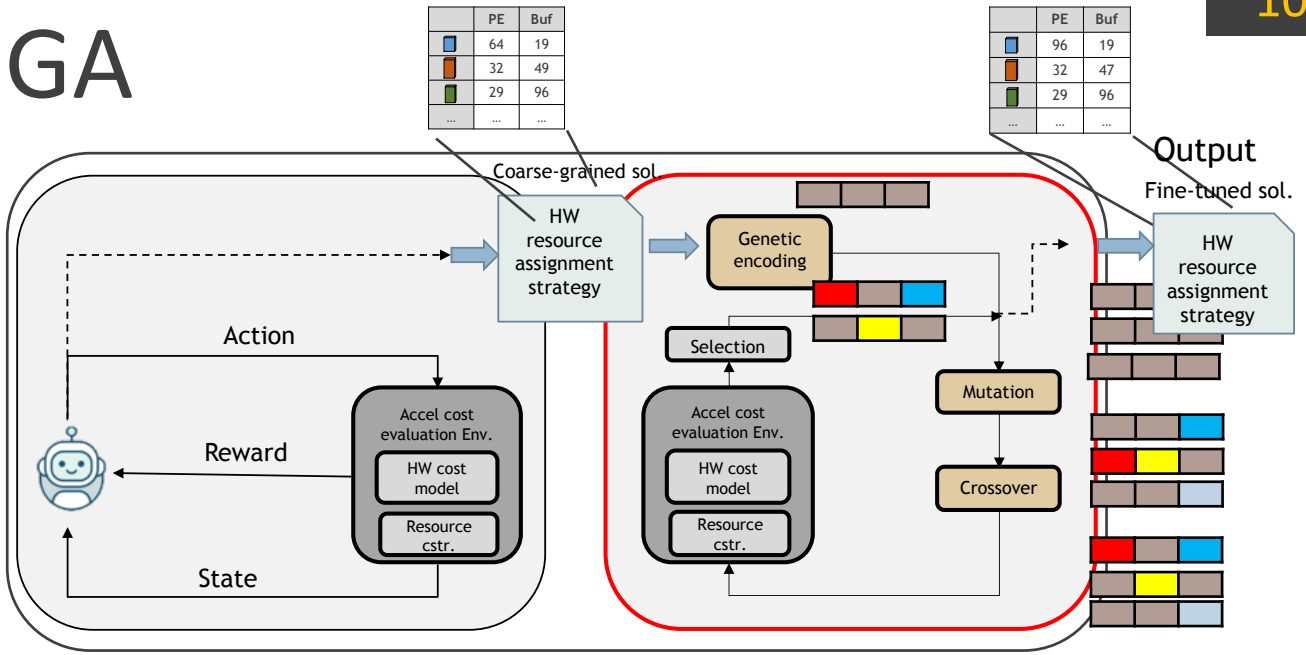
	PE	Buf	Reward
	3	6	20
	12	9	22
	5	3	15

Accumulated Reward : 57

Local Fine-tuning using GA

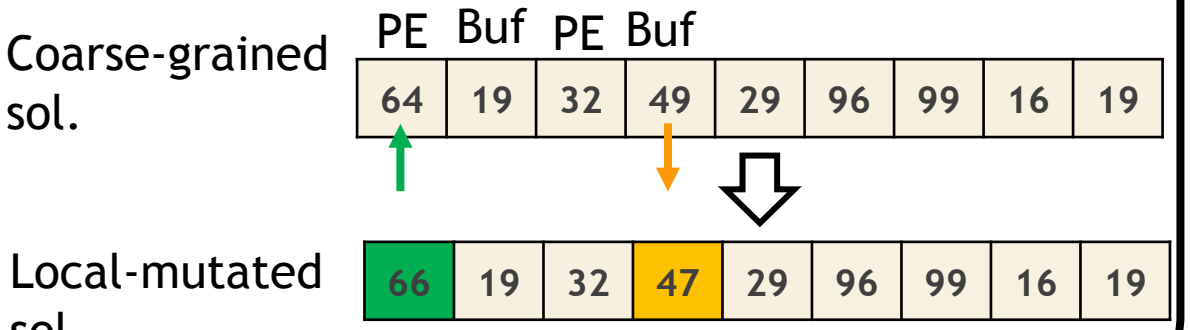
GA action space

- ± step size
- E.g., step size=4, PEs=64, action space=[60, 68]

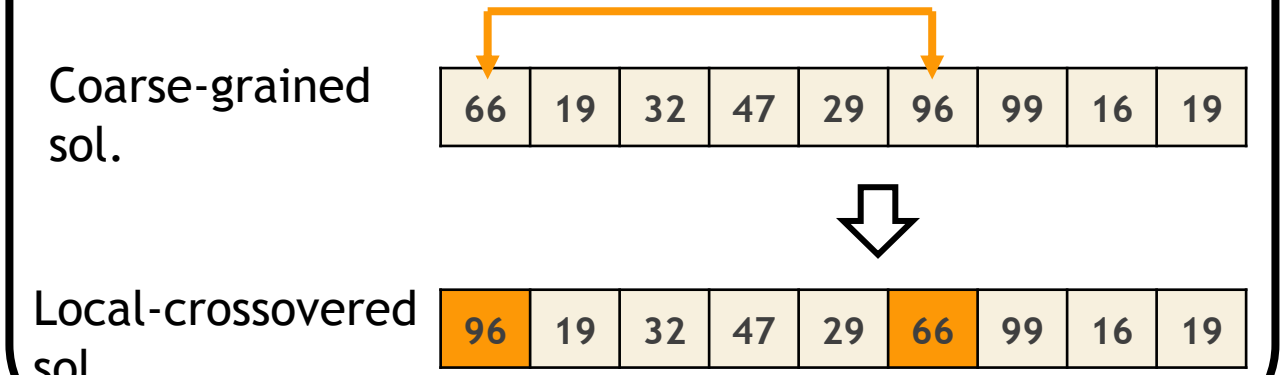


Genetic Operators

Local mutation



Local crossover



More details in the paper.

ConfuciuX Setup

- Setup
  - git clone https://github.com/maestro-project/confuciux.git
  - conda activate gammaEnv
  - pip install -r requirements.txt
  - python build.py

- Run ConfuciuX
  - sh ./run_ConfX.sh

- Run optimization methods
  - sh ./run_otherOpts.sh

- Run RL methods
  - sh ./run_otherRLs.sh

ConfuciuX Code-base

User will select one of them from the argument

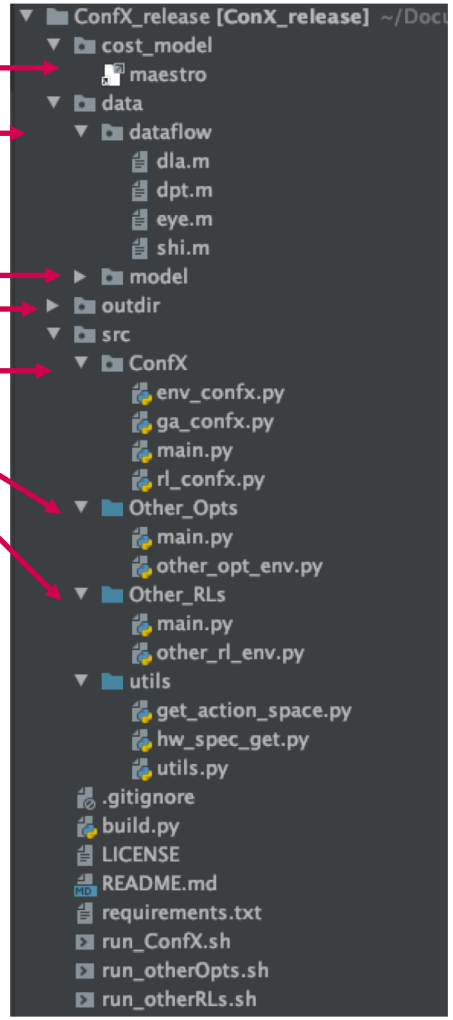
Soft-link to maestro

Supported Mapping/dataflow

Supported model descriptions

Result directory

Main program



./data/dataflow/dla.m

Change the buffer size by changing tile size of K.
(Here is where your action for Buffer apply.)

```
Dataflow {
    SpatialMap(KTileSz,KTileSz) K;
    TemporalMap(ClusterSz,ClusterSz) C;
    TemporalMap(Sz(R),Sz(R)) R;
    TemporalMap(Sz(S),Sz(S)) S;
    TemporalMap(Sz(R),1) Y;
    TemporalMap(Sz(S),1) X;
    Cluster(ClusterSz, P);
    SpatialMap(1,1) C;
    TemporalMap(Sz(R),1) Y;
    TemporalMap(Sz(S),1) X;
    TemporalMap(Sz(R),Sz(R)) R;
    TemporalMap(Sz(S),Sz(S)) S;
}
```

(However, it is not required to change by the K dim., you can use your own dataflow file to explore Buffer with different tile dimension.)

User Options

python main.py --outdir outdir --model example --fitness latency --cstr area --mul 0.5 --epochs 500 --df shi --alg RL_GA

Objective

* fitness: The fitness objective (latency/ energy)
* df: The dataflow strategy
* model: The model to run (available model in model dir)

Constraint

* cstr: Constraint (area/ power)
* mul: Resource multiplier. The resource ratio, the design is allowed to use.

*The system under design is only allowed to use mul * power_max or mul * area_max.*

Hyper parameters

* epochs: Number of generation for the optimization
* alg: The algorithm to run
  * For ConX, choose from [RL, RL_GA]
  * For RL, choose from [PPO2, A2C, ACKTR, SAC, TD3, DDPG]
  * For optimization methods, choose from [genetic, random, bayesian, anneal, exhaustive]
* outdir: The output result directory

Learning rate, discount ratio, mutation rate, etc., can be set at codebase

User-defined Action Space

Inside ./src/utils/get_action_space.py

Define your action space here

```python
import numpy as np
def get_action_space( use_default=True):
    if use_default:
        action_size = 12
        action_space = [np.array([1, 2, 4, 8, 12, 16, 24, 32, 48, 64, 96, 128]),                # Choice for number of PEs
                        np.array([i + 1 for i in range(action_size)]), ]                          # Choice for number of buffer unit

    else:


        raise NameError("Please define the customized action space.")
```

[[The choice for # of PEs]
[The choice for buffer sizes]]

[Advanced User] ConfuciuX-specific Tuning

RL: In ./src/ConfX/rl_confx.py

```
LR_ACTOR = 1e-3 # learning rate of the actor
GAMMA = 0.9  # discount factor
CLIPPING_LSTM = 10
CLIPPING_MODEL = 100
EPISIOLON = 2**(-12)
```

RL-specific hyper-parameters

Tune number of generation/populations

GA: In ./src/ConfX/main.py

```
def genetic_search(best_sol, best_reward, action_bound, action_bottom, num_layers=None, num_generations=100, num_pop = 20):
```

In ./src/ConfX/ga_confx.py

Mutation ratio.

```
def self_crossover(pop, eps=0):
```

[Advanced User] Add your Own Opt. Algorithm

In ./src/Other_Opts/main.py

```
if method == "random":
    env.random_search(opt.epochs, chkpt_file)
elif method == "exhaustive":
    env.exhaustive_search(opt.epochs, chkpt_file = chkpt_file, stride = opt.stride)
elif method == "genetic":

    env.genetic_search(epochs = opt.epochs, chkpt_file=chkpt_file)

elif method == "bayesian":
    env.bayesian_search(opt.epochs, chkpt_file=chkpt_file)

elif method == "anneal":
    env.anealing_search(opt.epochs, chkpt_file=chkpt_file)

else:
    print("Please choose from [genetic, random, bayesian, anneal, exhaustive]")
    exit(-1)
```

Add your algorithm here

In ./src/Other_Opts/other_opt_env.py

Random search method

```
for epoch in range(max_epoch):
    self.epoch = epoch
    guess_action = []
    for _ in range(n_layer):
        self.start_range = start_range
        self.end_range = end_range-1
        pe = action_space[0][random.randint(self.start_range, self.end_range)]
        bf = action_space[1][random.randint(self.start_range, self.end_range)]

        action = [pe, bf]
        guess_action.append(action)

    reward, total_used_constraint = self.exterior_search(guess_action)
```

You could use the same code structure.
The only work for your algorithm is to suggest
[PE, Buffer] for each layer

Resources

Paper:

S.-C. Kao, G Jeong, T Krishna, "ConfuciuX: Autonomous Hardware Resource Assignment for DNN Accelerators using Reinforcement Learning", MICRO, 2020

Repo:

https://github.com/maestro-project/confuciux